ISI Research Report
ISI/RR-91-282
April 1991

2

University of Southern California

Tzyh-Yung Wuu

Netlist+: A Simple Interface Language for Chip Design

INFORMATION SCIENCES INSTITUTE ISI
213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292 6695

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS —
2a. SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	3. DISTRIBUTION/AVAILABILITY OF REPORT This document is approved for public release: distribution is unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ISI/RR-91-282	5. MONITORING ORGANIZATION REPORT NUMBER(S) ———
6a. NAME OF PERFORMING ORGANIZATION USC/Information Sciences Institute 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION ———
6c. ADDRESS (City, State, and ZIP Code) 4676 Admiralty Way Marina del Rey, CA 90292-6695	7b. ADDRESS (City, State, and ZIP Code) ———
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F29601-87-C-0069
8c. ADDRESS (City, State, and ZIP Code) DARPA 1400 Wilson Boulevard Arlington, VA 22209	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. ——— PROJECT NO. ——— TASK NO. ——— WORK UNIT ACCESSION NO. ———

11 TITLE (Include Security Classification)
NetList+: A Simple Interface Language for Chip Design

12. PERSONAL AUTHOR(S)
Wuu, Tzyh-Yung

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
	FROM ____ TO ____	1991, April	79

16. SUPPLEMENTARY NOTATION

17 COSATI CODES

FIELD	GROUP	SUB-GROUP
09	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
ASIC, CAD tools, cell library, circuit, netlist, parser/generator, PLA, place and route (P&R), prototype, RAM/ROM, simulation, specification, standard cells, VLSI

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This describes a simple circuit specification language, NetList+, which was developed at MOSIS for rapid turn-around cell-based ASIC prototyping. By using NetList+, a uniform representation is achieved for the specification, simulation and physical description of a design. Our goal is to establish an easy interfacing method between design specification and independent CAD tools so that a simple description can be used for various tools.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Christine Tomovich	22b. TELEPHONE (Include Area Code) 213/822-1511 — 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

ISI Research Report
ISI/RR-91-282
April 1991

University of Southern California

Tzyh-Yung Wuu

Netlist+: A Simple Interface Language for Chip Design

INFORMATION SCIENCES INSTITUTE ISI
213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

This research was sponsored in part by the Defense Advanced Research Projects Agency under Contract F29601-87-C-0069. Views and conclusions contained in this report are the author's and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, or any person or agency connected with them

ACKNOWLEDGEMENTS

I would like to thank my supervisor, friend and colleague, Shih-Lien Lu for his encouragement and many helpful discussions. He provided special insight and valuable suggestions for this research. I also would like to thank Jen-I Pi who fixed bugs in public domain tools which I used. Without his enthusiasm, I would not have the perfect environment for this system. He also gave valuable suggestions to enhance the features of this system. I also want to thank David Tornheim who not only corrected my writing errors but provided critiques and comments which sharpened the ideas of this report. I also want to thank Dr. Ronald Ayres for many helpful discussions about the FUSION SERVICE. I also want to thank the following organizations for their CAD tools: Microelectronics Center of North Carolina for the VPNR place and route system, Northwest LIS of University of Washington for their NETLIST and RNL, University of California Berkeley for their MAGIC and related tools. Finally, I would like to thank my MOSIS colleagues for their encouragement and friendship. This research was supported in part by the Defense Advanced Research Projects Agency under Contract F29601-87-C-0069.

Contents

List of Figures

1 Introduction

NetList$^+$ is a design specification language developed at MOSIS for rapid turn-around cell-based ASIC prototyping. By using NetList$^+$, a uniform representation is achieved for the specification, simulation, and physical description of a design. Our goal is to establish an interfacing methodology between design specification and independent CAD tools. Designers need only to specify a system by writing a corresponding netlist. This netlist is used for both functional simulation and timing simulation. The same netlist is also used to drive the low level physical tools to generate layout. Another goal of using NetList$^+$ is to generate parts of a design by running it through different kinds of placement and routing (P&R) tools. For example some parts of a design will be generated by standard cell P&R tools. Other parts may be generated by a layout tiler — i.e. datapath compiler, RAM/ROM generator, or PLA generator. Finally all different parts of a design can be integrated by general block P&R tools as a single chip. The NetList$^+$ language can actually act as an interface among tools, and the relation between NetList$^+$ language and other tools is shown in Fig. 1. Currently NetList$^+$ is linked to simulation tools RNL [9] and IRSIM [3, 10], which are switch-level simulators, and SPICE [8] via a LISP-like network description language, NETLIST [9]. It is also linked to the P&R tools VPNR [6], which is a standard cell P&R tool, and FUSION [1, 2], which is a general block P&R tool. Other tools will be linked to NetList$^+$ in the near future.

In section 2 we will use a flowchart to illustrate the NetList$^+$ system and its relation with other related design tools. In section 3 we will show how to write a NetList$^+$ description from the block diagram of a circuit. In section 4 we will discuss how to prepare a cell library or several cell libraries for a design system. In section 5 we will give a few designs by NetList$^+$ and show their simulation and layout results in the appendix. Finally we will show you the ASIC design environment supported by NetList$^+$. In that section an example circuit is composed of random logic from VPNR, user-defined cells, and some function blocks from module generators.

2 NetList$^+$ System Organization

The original purpose of having NetList$^+$ language is to enable designers to write a simple description for multiple usages. There are three main parts in this system. In order to talk to other tools in different languages, we developed parser/generators to transform the NetList$^+$ language to an appropriate representation. In order to compile with cell libraries for different tools, we developed a series of programs to transform each cell from physical layout to the cell representation for different tools. Then we organized those cells in a file system so that programs in the design system can search them easily. Fig. 2 illustrates the NetList$^+$ system and related design tools. We use the convention of dotted extension

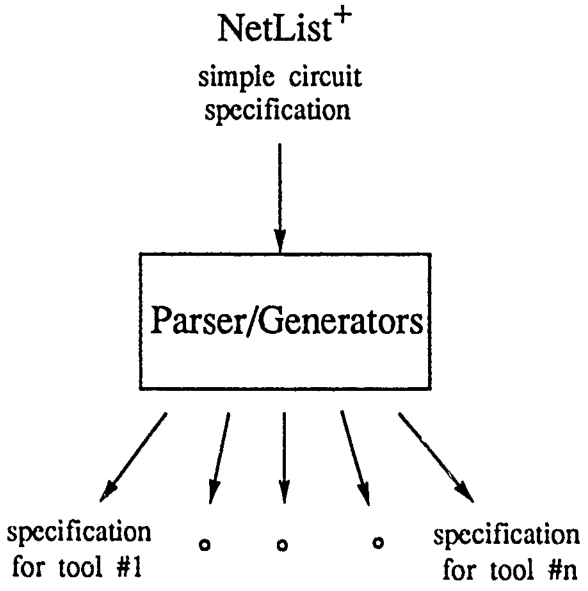
NetList+
simple circuit specification
Parser/Generators
specification for tool #1
o o o
specification for tool #n

Figure 1: NetList$^+$ Interfacing Methodology

names to represent the input/output files for each tool. These dotted extension names are:

name.fnet	NetList$^+$ netlist;
v_*name*.net	NETLIST for VPNR [6];
name.net	NETLIST for RNL/IRSIM[9];
name.spc	FUSION_SPEC[1];
name.lib	FUSION library[1];
name.cf	FUSION .cf file[1];
name.fu	FUSION file[1];
name.fused	fused file after FUSION[1];
cellname.cif	CIF file[7];
cellname.mag	MAGIC file[4];
cellname.ext	MAGIC .ext file[4];
cellname.sim	SIM file[4];
cellname.lif	FUSION leaf cell[1];
cellname.celi	NETLIST macro cell for RNL/IRSIM;
cellname.vpnr	NETLIST macro cell for VPNR;
cellib.db	VPNR .db file [6];

We will use this notation throughout this report.

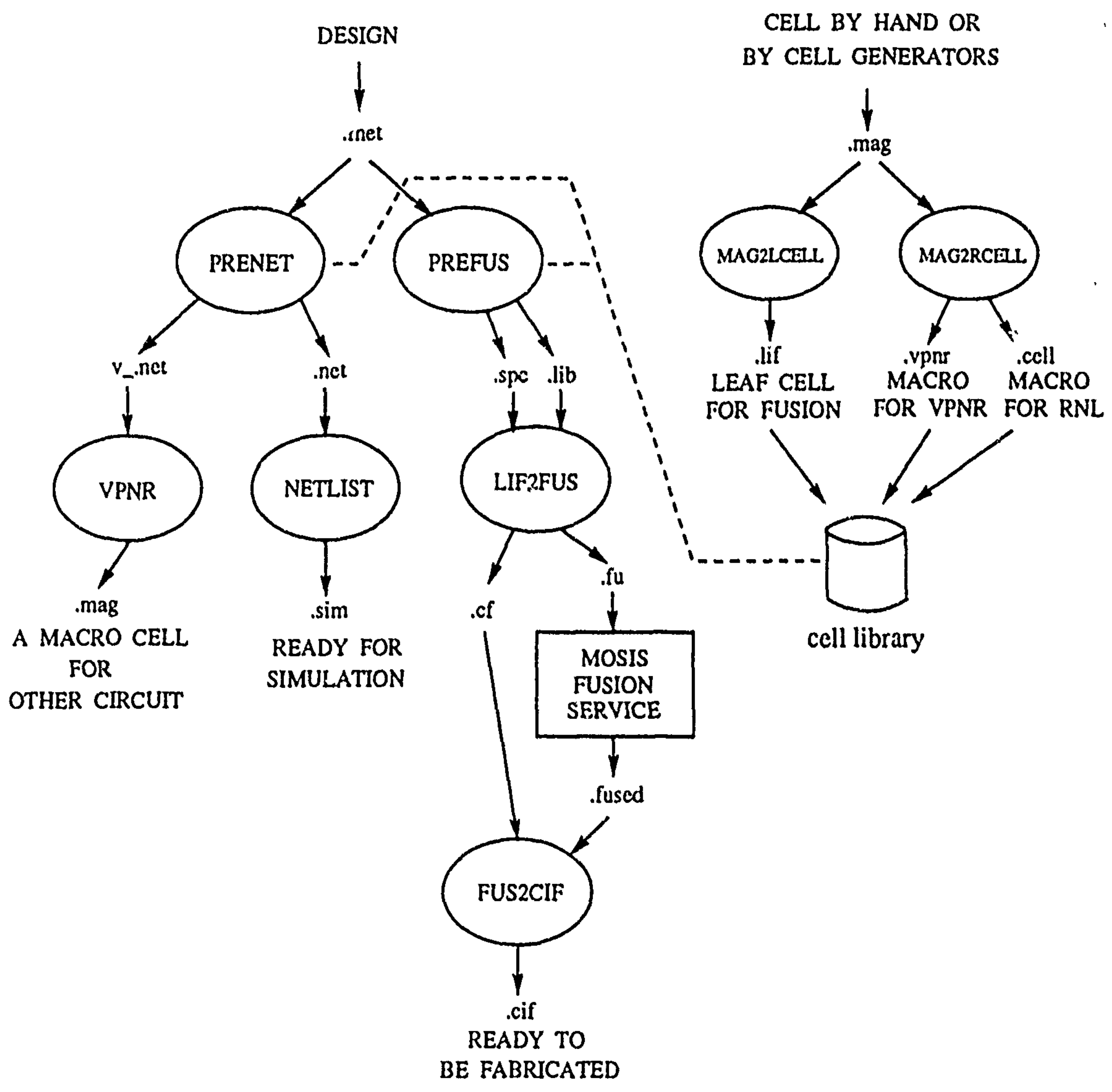

Figure 2: NetList+ system and Related Tools

3 NetList$^+$ and Circuit Design

NetList$^+$ is a LISP-like circuit design language. It is based on the NETLIST of NW Laboratory for Integrated Systems (LIS) [9]. We omitted the low level circuit descriptions like transistors, capacitors, resistors, etc. Besides the original functions and macros of NETLIST, there are descriptions to specify the directories of cell libraries, descriptions to indicate the inputs and outputs of the design, and functions to describe the floorplan of the circuit in NetList$^+$. These extra functions help the parser/generators to search cells, to generate input and output specifications, and to floorplan the structure for P&R tools.

Basically, we treat a digital system as a composition of blocks and nets among blocks. Each block is either a composition of small blocks or a well-defined cell. Here a well-defined cell means the cell has been layouted in MAGIC [4] or generated by some other macro generators. We assume NetList$^+$ users have one or more cell libraries already in MAGIC layout form in their systems. For each cell in a library, we can build a macro for simulation and a leaf cell for P&R. We have programs to generate the macro and leaf from *MAGIC* layout. We will describe the details of the preparation of cell libraries in the next section. W' ~n the cell libraries are ready, users can compose a circuit they want in NetList$^+$. Then users can user the description in NetList$^+$ to generate NETLIST for simulation, to generate a VPNR netlist for standard cell P&R, or to generate s FUSION specification for block P&R. In the following paragraph we discuss functions currently supported by NetList$^+$. Examples are used to illustrate the usage of NetList$^+$. In the rest of this report, keywords used by NetList$^+$ are shown in "**bold**" form in all our examples.

NETLIST Original Functions NetList$^+$ inherits most features of NETLIST. The basic elements of a circuit for NETLIST are cells and nets. All nets should be declared before they are used. The syntax of the net declaration is

(**node** *list_of_nets*)

The *list_of_nets* is a list of net names which are used in the circuit. The primitive description of a circuit in NetList$^+$ is a cell. The syntax of a cell is

(*cellname net_list*)

In fact, each *cellname* is a macro of NETLIST, which is from either a cell library or user-defined macro. The macro from a cell library encapsulates detailed transistor, resistor, and capacitor information. The user only invokes a cell by *cellname*. The other macro is composed of other cells. If some ports of a block are floating, we simply put "*" on the corresponding port when the cell is invoked. NetList$^+$ also support the constructs, **macro**, **repeat**, **connect**, **load**, **setq**, and LISP arithmectic operators from NETLIST. Any context after ";" till end of line is comment. The details about functions and macros of NetList+ can be found in appendix E and the original reference [9].

Example: If we want to design a circuit R = A·B + D·E as shown in Fig. 3, then the corresponding netlist in NetList+ can be written as follows:

```
(node A B C D E F R)      ; declaration of nodes
(AND A B C)
(AND D E F)
(OR C F R)
```

Clearly, it is straight forward to describe a circuit in NETLIST format.

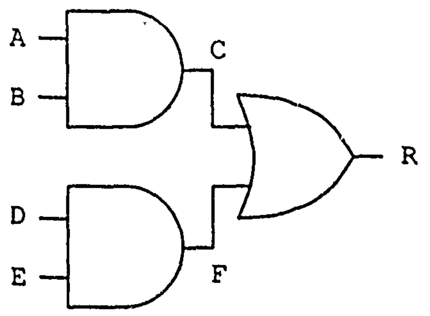

Figure 3: Logic Diagram for R = A·B + D·E

Library Path Declaration In the previous example, there are two AND gates and one OR gate. But we need to let the parser/generators know where to search for those cells. Hence we need to declare the cell library path. The syntax of the library path declaration is

(libpath "*path_of_cell_library*")

The *path_of_cell_library* is either a relative pathname or a full pathname. Multiple libraries are allowed in a circuit, and the parser/generator will search the cell according to the order of the library path declaration. If the logic gates are in directory /usr/mylib, we can add the command

```
(libpath "/usr/mylib")
```

at beginning of the netlist.

Input/Output Declaration Now we want to express the input/output of the design so that P&R tools can connect them to the pads of a chip or generate the I/O ports in some specific side of the module. The syntax of the I/O declaration is

(*side_of_block* (*type_of_io io_pin_list*))

The *side_of_module* is optional for P&R tools, and it can be either **east**, **west**, **south**, or **north** to indicate the side of block. The *type_of_io* is either **in**, **out**, or **io** to denote the type of the following signal list. The *io_pin_list* is a list of I/O pins. Each of them can be a single net name or a name with dimensions. The signal with dimensions is declared as follows:

(**singals** *start_index stop_index io_pin_list*)

For example, there is a 8-bit input bus, named INBUS, of block. We can simply declare the input pins as

(**in** (**signals** 0 7 INBUS))

In Fig. 3, if A, B, D, E are input pins and R is an output pins, we need to add

```
(in A B D E)
(out R)
```

in the netlist. If you want to run VPNR after NetList$^+$, you have to specify the side for each input and output pins, for example,

```
(north (in A B D E))
(south (out R))
```

This indicates that A, B, C, D are inputs going in the circuit from the *north*, and R is an output leaving the circuit from the *south* side.

Floorplan Specification Since some P&R tool like FUSION allow users to specify their own floorplan, we support floorplan specification in NetList$^+$. This function is optional for those designers who want to specify the floorplan of cells. The syntax of the floorplan is

(*orient cells*)

The *orient* is either **up**, **down**, **right**, or **left** to denote the orientation of the *cells* arranged on the layout. The *cells* can be a set of blocks, and each block may be another oriented cell. Also *cells* may be constructed by **repeat**. If we want the floorplan of the previous example shown in Fig. 4, then the netlist is written as,

```
(libpath "/usr/mylib")
(node A B C D E F R)
```

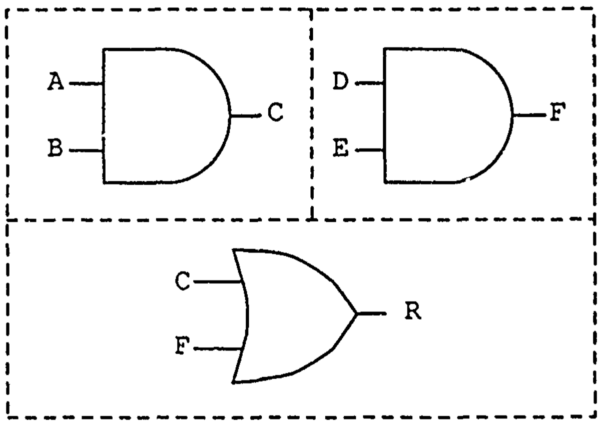

Figure 4: Floorplan for A·B, D·E, and C+F

```
(north (in A B D E))
(south (out R))
(down
   (right
      (AND A B C)
      (AND D E F)
   )
   (OR C F R)
)
```

The floorplan specification is useful for regular-structure designs like systolic arrays, especially used with the function **repeat**.

After the circuit description in NetList$^+$ is ready, we can run the design through **prenet**, **prefus** to get NETLIST, VPNR netlist and FUSION specification. There are more examples including simulation and layout in a later section.

4 Preparation of Cell Libraries

The original purpose of having the NetList$^+$ language is to enable designers to write a simple description for multiple usages. Since each tool has its own specification, we developed a series of programs to transform each cell from physical layout to the cell representation for different tools. The relationship between those files is shown in Fig. 2. NETLIST-related tools [9], FUSION [1, 2] and VPNR [6] are the first three tools linked to NetList$^+$ at MOSIS. Currently, we only support MAGIC [4] based layout, which means we assume designers use

MAGIC to generate all leaf cells in libraries. In this section we will discuss how to generate different representations for each tool and how to arrange different representations in the file system. The later is related to the library search in **prenet** and **prefus**.

4.1 NETLIST Macro Cell and Fusion Leaf Generation

For each cell there is a MAGIC file which describes the physical layout of the circuit. We need to build a macro for simulation which corresponds to the physical layout of the cell. We also need to build a leaf cell for FUSION which describes the geometry of the layout. Since macro and leaf are generated in the same way for each cell, we wrote a program and several utility scripts to take care of the jobs.

MAGIC has a command **extract** for extracting the circuit topology from layout. There is a command **ext2sim** to transform the .ext file to .sim file [4]. We have a program, **sim2cell**, to generate the NETLIST macro .cell file from a .sim file. For convenience' sake, we wrote a script, **mag2rcell**, which combines MAGIC's **extract**, **ext2sim**, and **sim2cell** to generate the .cell from a .mag directly. Since VPNR's macro has the same form as RNL's, **sim2cell** can generate the .vpnr file at the same time. We only need to put the option "-v vpnr_dir" in the command **sim2cell**.

Example Assume theie is a *mycell*.mag in $mylib/mag. If the current default directory is $mylib/rnl, we want to generate *mycell*.cell and *mycell*.vpnr in $mylib/rnl and $mylib/vpnr respectiverly. We can use the command:

$mag2rcell -m ../mag -v ../vpnr *mycell*

Example Assume we have .mag files for the cells *mycell1*, *mycell2*, *mycell3*, and *mycell4* in $mylib/mag. We can write a simple *makefile* stored in the ../rnl directory as follows to generate and maintain all .cell files and .vpnr files :

```
L = ../mag

OBJ1 = mycell1.cell mycell2.cell mycell3.cell mycell4.cell
OBJ2 = mycell1.head mycell2.head mycell3.head mycell4.head

all: $(OBJ1) $(OBJ2)

mycell1.lif: $(L)/mycell1.mag
    mag2rcell -m ../mag -v ../vpnr mycell1

mycell2.lif: $(L)/mycell2.mag
    mag2rcell -m ../mag -v ../vpnr mycell2

mycell3.lif: $(L)/mycell3.mag
```

```
    mag2rcell -m ../mag -v ../vpnr mycell3

mycell4.lif: $(L)/mycell4.mag
    mag2rcell -m ../mag -v ../vpnr mycell4
```

Most P&R tools have programs to generate a leaf file from a MAGIC file for each cell, so we may write scripts for building the leaf file for each cell. We also have a script, mag2lcell, which generate a FUSION .lif file from a .mag file.

Example For the same cell library in previous example, we want to generate and maintain the .lif's for FUSION. We can also have a *makefile* in $mylib/fusion such as,

```
L = ../mag
B = ../rnl
OBJ1 = mycell1.lif mycell2.lif mycell3.lif mycell4.lif master.lib
OBJ2 = $(B)/mycell1.head $(B)/mycell2.head $(B)/mycell3.head $(B)/mycell4.head

all: $(OBJ1) master.lib

master.lib: $(OBJ2)
    masterlib ../rnl master.lib

mycell1.lif: $(L)/mycell1.mag
    mag2lcell -m ../mag mycell1

mycell2.lif: $(L)/mycell2.mag
    mag2lcell -m ../mag mycell2

mycell3.lif: $(L)/mycell3.mag
    mag2lcell -m ../mag mycell3

mycell4.lif: $(L)/mycell4.mag
    mag2rcell -m ../mag mycell4
```

Currently, we also manually maintain VPNR .db files. If you want to know the details of the specifications for FUSION and VPNR, you may refer to the corresponding user manual [1, 6]. More detailed descriptions for commands used above are shown in the manual pages.

4.2 File System for Cell Libraries

In order to have a uniform search path for cell libraries, we only accept two kinds of file structure as the internal representation for NetList$^+$ related programs. The first is used to put all MAGIC cells, RNL macro cells, FUSION leaf cells, and VPNR leaf cells of a cell library in the same directory. The other method is to have one directory for the cell library where MAGIC cells, RNL macro cells, FUSION leaf cells, and VPNR leaf cells are placed in the subdirectory *mag*, *rnl*, *fusion*, and *vpnr* respectively. The former method is suitable for a small cell library, and the latter is used for larger cell library. We suggest designers use the second.

We give examples to demonstrate these two kinds of file systems. Consider a cell library containing five cells, INV, NAND2, NOR2, XOR, and FADD. The cells and their internal representation can be stored in a single directory, and the file system will appear as in Fig. 5(a). On the other hand we can put them in several subdirectories as is shown in Fig. 5(b).

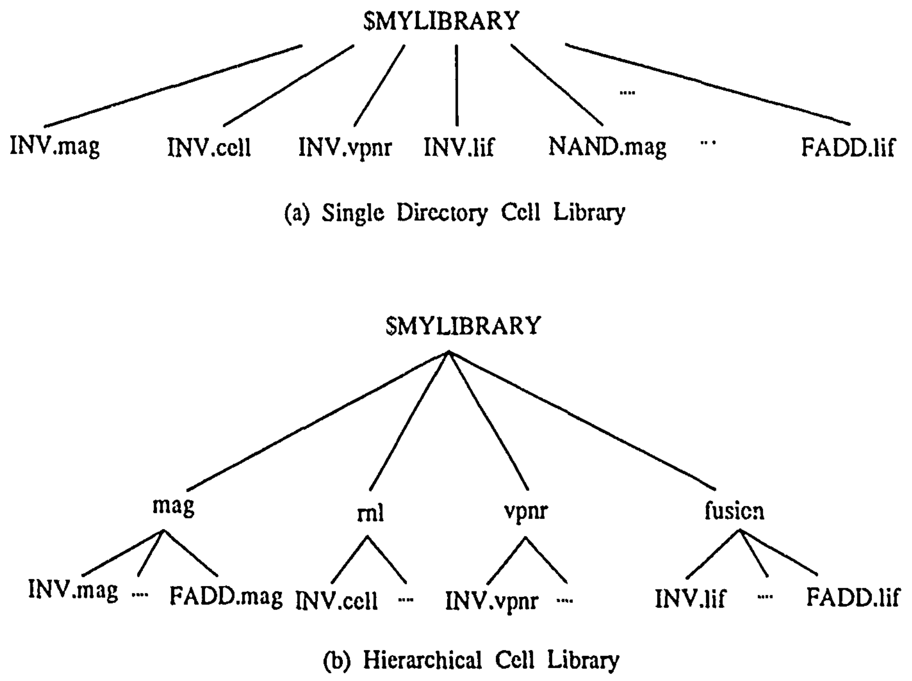

Figure 5: File System for Cell Library

Finally, we will talk about the relation between library declarations and the program searching order in NetList$^+$. In **prenet** and **prefus**, the order of cell searching is current directory, subdirectory[1] of current directory, directory of the first declared library, subdirectory of the first declared library, directory of the second declared library, etc.

Example In the example R = A·B + D·E, if there is a cell AND.cell in the current directory, **prenet** will use ./AND.cell instead of /usr/mylib/AND.cell. If there is no

[1]subdirectory means respective subdirectory, e.g. ./rnl for an RNL macro cell.

cell ./AND.cell, then **prenet** will search for ./rnl/AND.cell, /usr/mylib/AND.cell, then /usr/mylib/rnl/AND.cell. If none of them exists, **prenet** will report the error.

5 Examples

In this section, we will show you several designs specified in NetList$^+$. Each example will demonstrate a different feature of NetList$^+$. In the first example, we will show you a netlist without any special structure. In the second example we will show you the loop structure in NetList$^+$, and we will also give a netlist with the structure to describe the relative location among cells. In the third example, we will show you the hierarchical design in NetList$^+$. In this example, we inherit the feature of NETLIST which allows a parameterized macro cell. In the last example, we have a complicated design. We design the circuit in two parts, datapath and control. These two parts are designed separately. After getting each part done, we compose them in one chip. All simulation and layout results for every design are shown in the appendix.

5.1 Example 1: 1-Bit Full Adder

In this example, we give you a flat design to show the relation between hardware design and netlist specification. Our first example is a gate-level design of 1-bit full adder as shown in

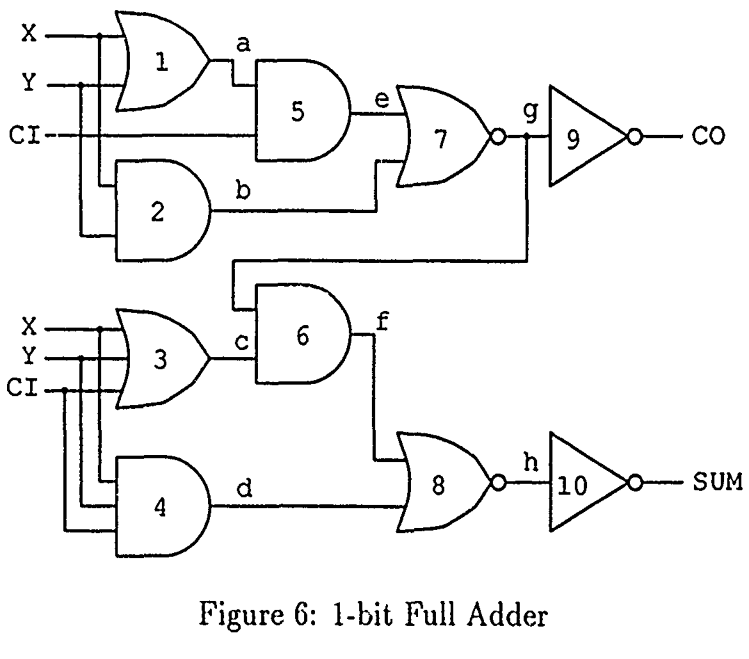

Figure 6: 1-bit Full Adder

Fig. 6. We used the "cmos/cmosn" library cells to build this circuit. The *cells* used in this

example are listed as follows:

Function	#inputs	Macro call
NOT	1	(inv2 IN OUT)
AND	2	(ndi2x1 IN1 IN2 OUT1 OUT2)
AND	3	(ndi3 IN1 IN2 IN3 OUT1 OUT2)
OR	2	(nri2 IN1 IN2 OUT1 OUT2)
OR	3	(nri3 IN1 IN2 IN3 OUT1 OUT2)
NOR	2	(nr2 IN1 IN2 OUT)

The netlist description of the circuit in Fig. 6 in NetList$^+$ is shown as follow:

```
(libpath "cmos/cmosn")

(node X Y CI SUM CO a b c d e f g h)

(north (in X Y CI))
(south (out SUM CO))

(nri2   X Y * a)              ; gate 1
(ndi2x1 X Y * b)              ; gate 2
(nri3   X Y CI * c)           ; gate 3
(ndi3   X Y CI * d *)         ; gate 4
(ndi2x1 a CI * e)             ; gate 5
(ndi2x1 g c * f)              ; gate 6
(nr2    e b g)                ; gate 7
(nr2    f d h)                ; gate 8
(inv2   g CO)                 ; gate 9
(inv2   h SUM)                ; gate 10
```

In this example, you may find the similarity between NetList$^+$ and NETLIST, except some description are introduced in NetList$^+$. *libpath* is used to specify the directory of the cell library. *in* and *out* are used to describe the input and output of the whole circuit, and *south* and *north* are used to specify the direction of input/output in the final layout. The "*" is used to indicate that a port has no connection. There is "*" in cell "nri2", which has four ports, IN1, IN2, OUT1, and OUT2. OUT1 is equal to the logic "nor" of IN1 and IN2, and OUT2 is equal to logic "or" of IN1 and IN2. Since OUT1 is not used in this circuit, we put a "*" on the port. Assuming that the above netlist is stored in file *fa*.fnet, we can run **prenet** by the command

```
$prenet -v fa
```

to obtain *fa*.net and v_*fa*.net, which are the NETLIST for RNL and the NETLIST for VPNR respectively. They can be used to run simulation and standard cell P&R. We can also run **prefus** by the command:

```
$prefus -plSLLU2_PADS -pf40PC fa
```

to obtain *fa*.spc and *fa*.lib, which are the FUSION netlists. They can be used to run FUSION, which is a block P&R with the ability to place & route pads. In the command, the option -pf and the option -pl specify the pad frame and pad library used in FUSION. There are several pad libraries and pad frames supported by FUSION, which are shown in appendix F. *fa*.net, *v_fa*.net, *fa*.spc, *fa*.lib, the simulation of RNL, the layout from VPNR, and the layout from FUSION are shown in appendix A.

5.2 Example 2: Parallel Multiplier

In this example, we use a design to show the loop structure and floorplan expression in NetList$^+$. Basically, the loop is inherited from NETLIST. We use the N-bit parallel multiplier to demonstrate the loop structure as well as floorplan expression. The diagrams

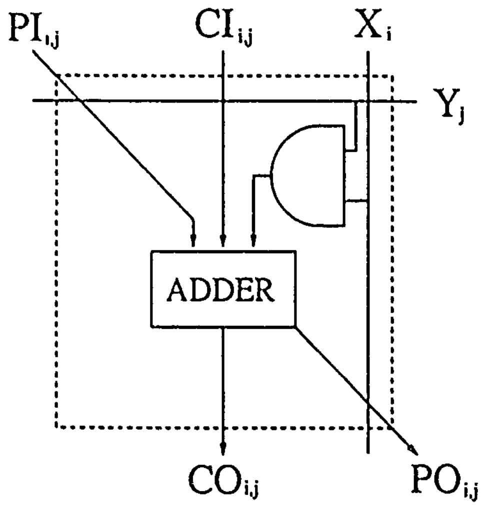

Figure 7: Parallel Multiplier Cell $M_{i,j}$

for the internal cell, the interconnection among cells, and 4-bit parallel multiplier are shown in Fig. 7, 8, and 9 respectively. We used the "cmos/cmosn" library cells to build this circuit. The *cells* used in this example are listed as follows:

Function	#inputs	Macro call
AND	2	(ndi2x1 IN1 IN2 OUT1 OUT2)
FADD	3	(fadd CO IN1 IN2 IN3 SUM FEED1 FEED2 FEED3)

Because negative index is not allowed in NETLIST, the index for input X, Y start from 1 to N and the index for output P starts from 1 to 2N. The index 0 in each dimension is

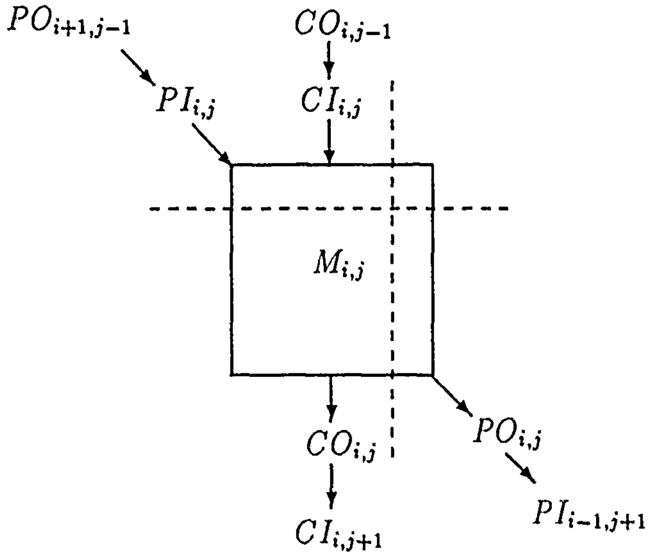

Figure 8: Parallel Multiplier Cell Connection

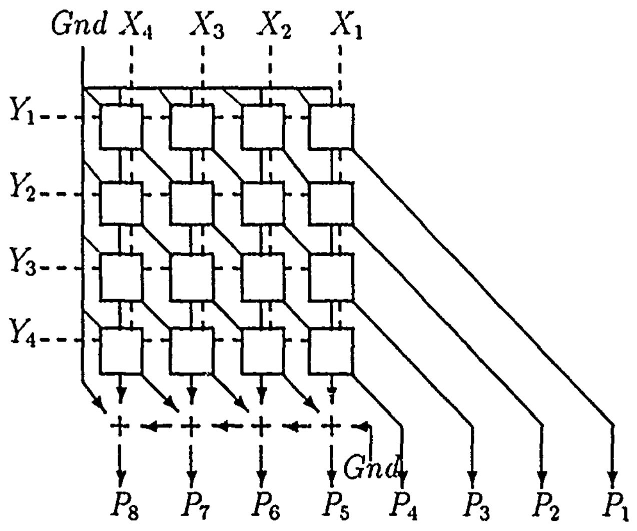

Figure 9: 4-bit Parallel Multiplier

reserved for boundary condition. The node names in netlist with respect to multiplier cell $M_{i,j}$ are denoted as follows (please refer to Fig. 8 and 9):

Net name	Corresponding port of cell $M_{i,j}$
X	N-bit input vector
Y	N-bit input vector
P	2N-bit output vector
$p.i.j$	$PI_{i,j}$ and $PO_{i+1,j-1}$
$c.i.j$	$CI_{i,j}$ and $CO_{i,j-1}$
$n.i.j$	internal connection between AND and FADD in $M_{i,j}$
$C.i$	Carry-in of FADD in column X_i

The netlist description for 4-bit parallel multiplier is shown as follows:

```
; N-bit parallel multiplier with CONNECT used to simplify
; the programming of NetList+

; Libraries inclusion and Global parameter setting
(setq N 4)
(libpath "cmos/cmosn")

; Node declaration and signals alias
(node X Y P n p c C)

(repeat i 1 N
  (connect p.i.1 GND)
  (connect c.i.1 GND)
)
(repeat j 2 (+ 1 N)
  (connect p.N.j GND)
)
(repeat j 1 N
  (connect p.0.(+ j 1) P.j)
)
(connect C.1 GND)

; I/O definitions
(north (in (signals 1 N X Y)))
(south (out (signals 1 (* 2 N) P)))

; Circuit Netlist
(repeat i 1 N
  (repeat j 1 N
    (ndi2x1 X.i Y.j * n.i.j)
    (fadd c.i.(+ j 1) p.i.j c.i.j n.i.j p.(- i 1).(+ j 1) * * *)
  )
)

(repeat i 1 N
  (fadd C.(+ i 1) C.i c.i.(+ N 1) p.i.(+ N 1) P.(+ i N) * * *)
)
```

In above example, NetList$^+$ inherited the *repeat* structure and the *connect* function from NETLIST. Due to the requirement of VPNR, the order of the description in NetList$^+$ is more restricted than NETLIST. The recommended order of description is

1. Libraries inclusion and global parameter setting.
2. Node declaration and signals alias.
3. I/O definitions.
4. Circuit netlist.

In fact, we use the order in above example. Assuming that the above netlist is stored in file *pm0*.fnet, we can run prenet by the command

```
$prenet -v pm0
```

to obtain *pm0*.net and v_*pm0*.net, which are the NETLIST for RNL and the NETLIST for VPNR respectively. They can be used to run simulation and standard cell P&R. We can also run prefus by the command

```
$prefus -l0.8 -plCMOSN16_PADS -pf40P -tSC_TWIN-TUB -i1 pm0
```

to obtain *pm0*.spc and *pm0*.lib, which are the netlist for FUSION SERVICE. In the command, the option -pf and the option -pl are described in the example 1, and the option -i specify the pad format for FUSION. The detailed explanation of the pad format is shown in appendix F. The option -l specifys the *lambda* value [7], and the option -t specifys the technology used. All the *pm0*.net, *v_pm0*.net, *pm0*.spc, *pm0*.lib and the simulation of RNL, the layout from VPNR, and the layout from FUSION are shown in appendix B.

Connect Though NetList$^+$ inherited a lot of functions and macros from NETLIST, not all of them work the same way as the RNL simulation. *Connect* is the macro working differently in RNL, VPNR and FUSION. In RNL, *connect* is used to specify a list of nodes electrically connected. In VPNR, *connect* is interpreted as the declaration of alias, and every node in the list will be replaced by the last node in the list. For example, after the statement

```
(connect a b c d)
```

node *a*, node *b*, and node *c* in the netlist will be replaced by node *d*. Now there is a case that is all right for RNL but would not work for VPNR. For example,

```
(connect a b)
(connect a c)
...
```

It means node *a*, *b*, *c* are connected in RNL, but there will have error message in VPNR. Because node *a* is replaced by node *b* in the first statement, no node *a* can be replaced by node *c* in the second statement. Thus

```
(connect a b)
(connect b c)
...
```

or

```
(connect a b)
(connect c b)
...
```

is acceptable statements in VPNR. Only those nodes which are not replaced by other nodes will be shown in final layout. In FUSION, there is no corresponding macro to *connect*, but we implement the *pseudo-connect* in FUSION. We actually build a cell *connect* shown in Fig. 10, so that the macro *connect* with n nodes in NetList$^+$ will become $n-1$ *connect* cells

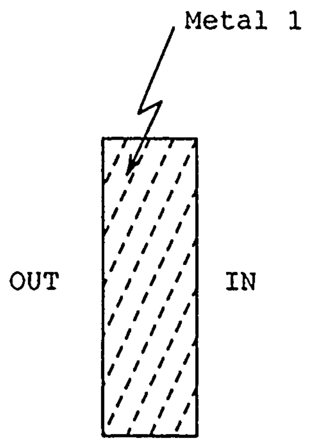

Figure 10: Cell *connect*

in FUSION. It is very convenient for the circuit designers to use *connect*, but it may have bigger layout for FUSION. In order to have smaller layout, we may need to avoid using *connect* in FUSION. For example, we can re-write the same circuit without any "connect" statement as follows:

```
; N-bit parallel multiplier without CONNECT used in
; the programming of NetList+
; Libraries inclusion and Global parameter setting
(setq N 4)
(libpath ''cmos/cmosn'')

; Node declaration and signals alias
(node X Y P n p c C)

; I/O definitions
(north (in (signals 1 N X Y)))
(south (out (signals 2 (+ N 1) p.0)
            (signals (+ N 1) (* 2 N) P)))

; Circuit Netlist
(repeat i 1 (- N 1)
  (ndi2x1 X.i Y.1 * n.i.1)
  (fadd c.i.2 GND GND n.i.1 p.(- i 1).2 * * *)
  (repeat j 2 N
    (ndi2x1 X.i Y.j * n.i.j)
    (fadd c.i.(+ j 1) p.i.j c.i.j n.i.j p.(- i 1).(+ j 1) * * *)
  )
)
(ndi2x1 X.N Y.1 * n.N.1)
(fadd c.N.2 GND GND n.N.1 p.(- N 1).2 * * *)
(repeat j 2 N
  (ndi2x1 X.N Y.j * n.N.j)
  (fadd c.N.(+ j 1) GND c.N.j n.N.j p.(- N 1).(+ j 1) * * *)
)

(fadd C.2 GND c.1.(+ N 1) p.1.(+ N 1) P.(+ 1 N) * * *)
(repeat i 2 (- N 1)
  (fadd C.(+ i 1) C.i c.i.(+ N 1) p.i.(+ N 1) P.(+ i N) * * *)
)
(fadd C.(+ N 1) C.N c.N.(+ N 1) GND P.(+ N N) * * *)
```

The layout from FUSION without *connect* are shown in appendix B.

Floorplan In FUSION, user-specified floorplan is allowed. Foe example, if we want the floorplan of our circuit as shown in Fig. 9. The netlist should be written as follows:

```
; N-bit parallel multiplier with floorplan used in
; the programming of NetList+

; Libraries inclusion and Global parameter setting
(setq N 4)
(libpath "cmos/cmosn")

; Node declaration and signals alias
(node X Y P n p c C)
```

```
; I/O definitions
(north (in (signals 1 N X Y)))
(south (out (signals 2 (+ N 1) p.0)
            (signals (+ N 1) (* 2 N) P)))

; Circuit Netlist
(down
  (left                         ; column 1~N are placed from right
                                ; 'to left'
    (repeat i 1 (- N 1)
      (down                     ; In each column, cells are placed
                                ; from top 'to down'
        (left                   ; fadd is on the left of ndi2x1
          (ndi2x1 X.i Y.1 * n.i.1)
          (fadd c.i.2 GND GND n.i.1 p.(- i 1).2 * * *)
        )
        (repeat j 2 N
          (left
            (ndi2x1 X.i Y.j * n.i.j)
            (fadd c.i.(+ j 1) p.i.j c.i.j n.i.j p.(- i 1).(+ j 1) * * *)
          )
        )
      )
    )
    (down
      (left
        (ndi2x1 X.N Y.1 * n.N.1)
        (fadd c.N.2 GND GND n.N.1 p.(- N 1).2 * * *)
      )
      (repeat j 2 N
        (left
          (ndi2x1 X.N Y.j * n.N.j)
          (fadd c.N.(+ j 1) GND c.N.j n.N.j p.(- N 1).(+ j 1) * * *)
        )
      )
    )
  )

  (left
    (fadd C.2 GND c.1.(+ N 1) p.1.(+ N 1) P.(+ 1 N) * * *)
    (repeat i 2 (- N 1)
      (fadd C.(+ i 1) C.i c.i.(+ N 1) p.i.(+ N 1) P.(+ i N) * * *)
    )
    (fadd C.(+ N 1) C.N c.N.(+ N 1) GND P.(+ N N) * * *)
  )
)
```

The "left" means the following blocks placed from right to *left*. For example,

```
..........
      (left
         (cella ....)
         (cellb ....)
         (cellc ....)
      )
..........
```

means cell *cella*, *cellb* and *cellc* are placed from right to left. Similarly, "right", "up", "down" means blocks *going right*, *going up* and *going down* respectively. The layout from FUSION with floorplan specified are also shown in appendix B.

5.3 Example 3: Pseudorandom Number Generator

We used this simple example to demonstrate the usage of parameterized macro definition in NetList$^+$. We wrote a macro of n-bit shift register, *nshift*, with parameter *n*. Then we used *nshift* to compose a pseudorandom number generator. The design of pseudorandom number generator is shown in Fig. 11 and Fig. 12. Again we use the "cmos/cmosn"

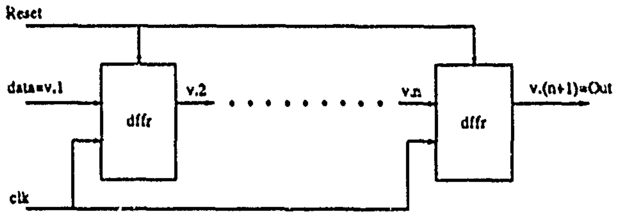

Figure 11: n-bit Shift Register

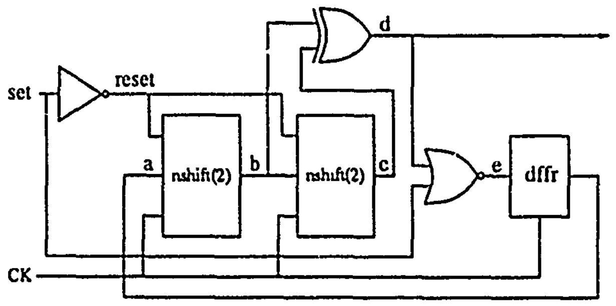

Figure 12: 5-bit Pseudorandom Number Generator

library cells to build this circuit. The cells used in this example are listed as follows:

Function	#inputs	Macro call
register w/ reset	3	(dffr CLOCK DATA Q QBAR RESET)
register w/o reset	2	(dff CLOCK DATA Q)
INV	1	(inv1 IN OUT)
NOR	2	(nri2 IN1 IN2 OUT1 OUT2)
XOR	2	(xor IN1 IN2 OUT)

The netlist description of the circuit in Fig. 12 in NetList$^+$ is shown as follows:

```
(libpath "cmos/cmosn")

(node a b c d e set reset CK)

(north (in CK set))
(south (out d))

(macro nshift ((int n)             ; macro definition of an n-bit
      data clk Out Reset) ; shift register
  (local v)
  (connect data v.1)
  (repeat i 1 n
    (dffr clk v.i v.(+ i 1) * Reset)
  )
  (connect Out v.(+ n 1))
)

(inv1 set reset)
(nshift 2 a CK b reset)
(nshift 2 b CK c reset)
(xor b c d)
(nri2 d set * e)
(dff CK e a *)
```

In the above example, *macro* definition is slightly different from the NETLIST. In NetList$^+$, users need to specify the type of parameters other than *node*. In this example, the **int** is used for parameterized macro definition. Another useful type is **signals**, which has the same meaning as the input/output declarations. The reason for type declaration is some P&R tools like FUSION support hierarchical design physically, that is, each macro cell will become a physical block in P&R. Thus preprocess like **prefus** need to decide the size of each macro cell send to P&R tools like FUSION. Other P&R tools like VPNR flatten all macro cells, so they do not need to specify the type of parameters. Currently, we only support macro cell for VPNR, and we will upgrade the macro for FUSION also. Due to the implementation of *connect* in VPNR, it is risky to use *connect* in macro in VPNR. Thus we replace the macro definition before sending to VPNR as follows:

```
(macro nshift (n data clk Out Reset)
  (local v float)
  (cond ((eq n 1) (dffr clk data Out float.1 Reset))
        (t (repeat i 1 n
                (cond ((eq 1 i) (dffr clk data v.2 float.1.i Reset))
                      ((eq n i) (dffr clk v.n  Out float.1.i Reset))
                      (t (dffr clk v.i v.(+ i 1) float.1.i Reset)))
                 )
            )
    ))
```

The *cond* structure from lisp is not supported in current version of NetList$^+$, and it will also be implemented in next version. The simulation from IRSIM and the layout from VPNR are shown in appendix C.

5.4 Example 4: Two's Complement Multiplier

In this example, we illustrate how a circuit can be composed of several blocks which come from different generators. We design a two's complement multiplier with the block diagram shown in Fig. 13 [5]. In the first two cycles, multiplier and multiplicand are read in from INBUS. And the following 16 cycles are doing "add" and "shift" operations. In the final two cycles, the 16-bit result is put on the OUTBUS. The sequence of the operations is shown in Fig. 14. There are two parts in this circuit. One is a datapath which calculates the data, and the other is a control unit which generates control signals. In order to avoid racing between control signals and latched data signals, the control signals are set at the rising clock edge, and the data are latched at the falling edge. We use the scalable CMOS library "cmos/scmos" to build the whole circuit. The *cells* used in this example are listed as follows:

Function	#input	Macro call
adder	3	(sc_fadd CO IN1 IN2 IN3 SUM F1 F2 F3)
multiplexer	4	(sc_mux1 IN1 IN2 IN3 IN4 OUT1 OUT2)
nand latch	2	(sc_ndlat Q QBAR R S)
nor latch	2	(sc_nrlat Q QBAR R S)
buffer	1	(sc_buff DATA OUT1 OUT2)
tri-state buffer	2	(sc_trib2 DATA EN OUT F1 F2 F3)
register w/ reset	3	(sc_dffr CLOCK DATA Q QBAR RESET)
register w/ set/reset	4	(sc_dffsr CLOCK DATA Q QBAR RESET SET)
8-bit shift register	14	(sc_sr8 CLOCK D0-D7 DL DR Q0-Q7 RESET S1 S2 F1-F19)
INV	1	(sc_inv2 IN1 OUT)
NAND	2	(sc_nd2x2 IN1 IN2 OUT)
NAND/AND	2	(sc_ndi2x1 IN1 IN2 OUT1 OUT2)
NAND/AND	3	(sc_ndi3 IN1 IN2 IN3 OUT1 OUT2 F1)
NOR	2	(sc_nr2 IN1 IN2 OUT)
NOR/OR	2	(sc_nri2 IN1 IN2 OUT1 OUT2)

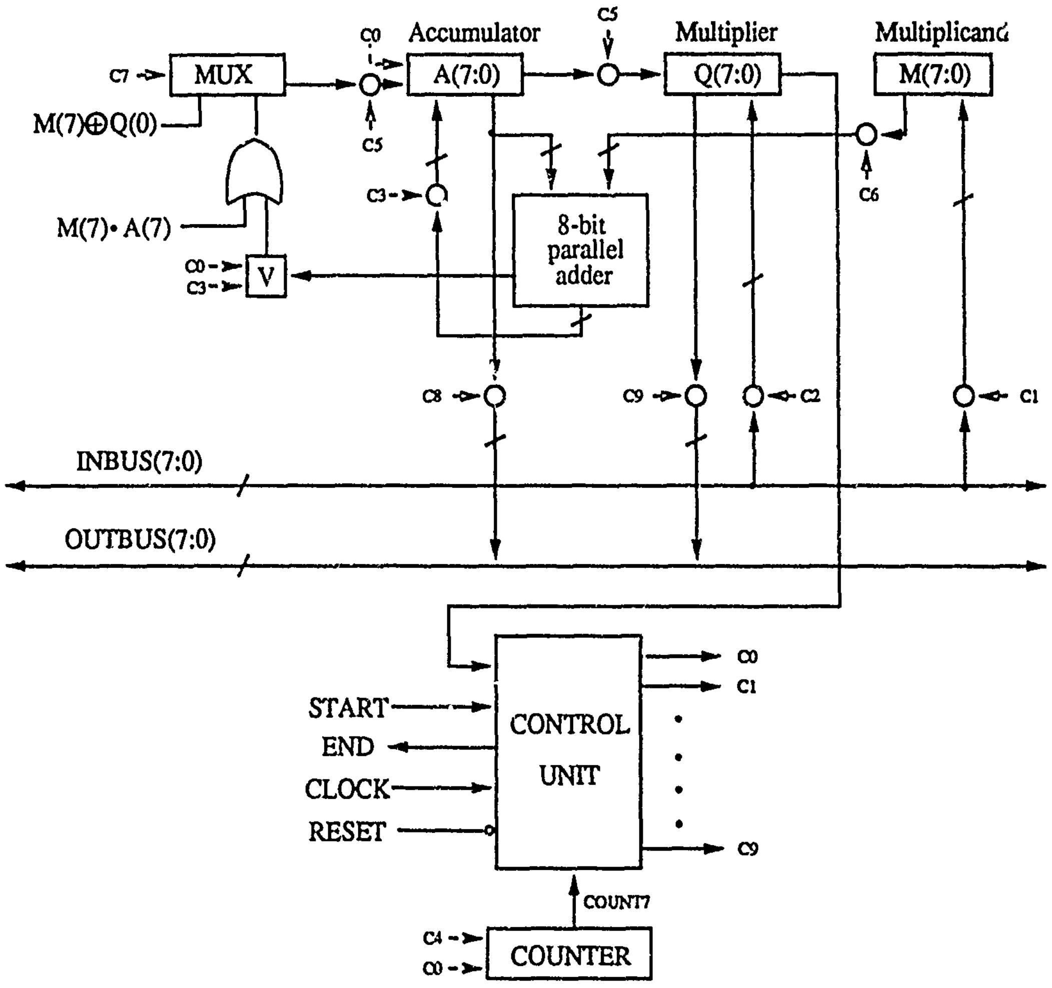

Figure 13: Block Diagram for Two's Complement Multiplier

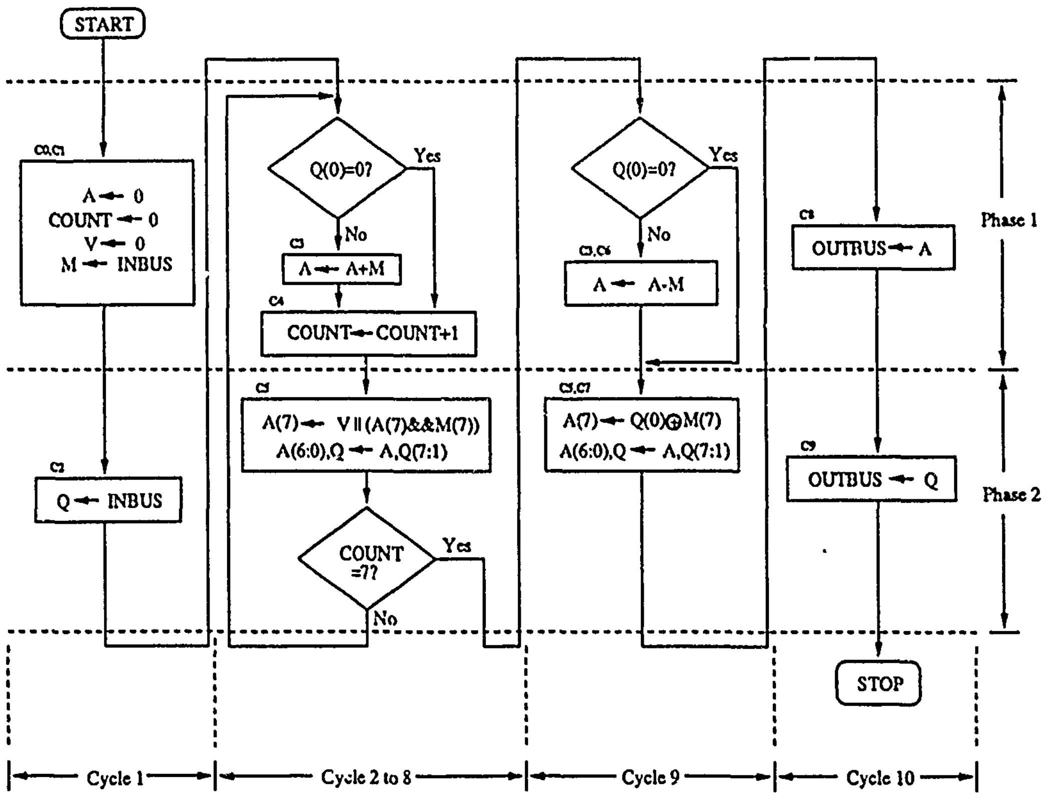

Figure 14: Flowchart for Two's Complement Multiplier

XOR	2	(sc_xor IN1 IN2 OUT)
XNOR	2	(sc_xnor IN1 IN2 OUT)

Datapath of Two's Complement Multiplier We use the VPNR to fuse all cells in the datapath together. According to the block diagram in Fig. 13, we have the corresponding netlist description for the datapath shown as follows:

```
; data path for 2's complement multiplier
; libpath declaration
(libpath "cmos/scmos")

; node declaration
(node INBUS OUTBUS)
(node c0 c1 c2 c3 c5 c6 c7 c8 c9 c0B c3B c7B CLK)
(node mout qin qout ain)
(node co x y s v vi vi0)
(node adl adl0 adl1 ma7)

; I/O declaration
(south (in c0 c1 c2 c3 c5 c6 c7 c8 c9))
(south (in CLK qout.0))
(north (out (signals 0 7 OUTBUS)))
(north (in (signals 0 7 INBUS)))

; macro declaration
; 8-bot adder
(macro adder8 (CI COUT IN1 IN2 SUM)
  (local cinter)
  (sc_fadd cinter.0 CI IN1.0 IN2.0 SUM.0 * * *)
  (repeat i 1 6
    (sc_fadd cinter.i cinter.(- i 1) IN1.i IN2.i SUM.i * * *)
  )
  (sc_fadd COUT cinter.6 IN1.7 IN2.7 SUM.7 * * *)
)

; 8-bit register w/ shift right
; reset, ld(load), sr(shift right) are active high
(macro dffsr8 (clock di qo dl ld sr reset)
  (local s1 s2)
  (sc_nr3 ld sr reset s2)
  (sc_nr2 ld reset s1)
  (repeat i 0 7
    (sc_gpuld reset di.i *)
  )
  (sc_sr8 clock di.0 di.1 di.2 di.3 di.4 di.5 di.6 di.7 dl *
          qo.0 qo.1 qo.2 qo.3 qo.4 qo.5 qo.6 qo.7
          Vdd s1 s2 * * * * * * * * * * * * * * * * * * * * *)
)

; 8-bit tri-state buffer
(macro tri8 (tin tout ten)
  (local float)
  (repeat i 0 7
    (sc_trib2 tin.i ten tout.i * * *)
  )
)
; 8-bit 2's complement
(macro comp8 (cai cao csel)
  (local it cit caoB)
  (sc_ndi2x1 csel GND * it.0)
  (sc_nri2  cai.0 GND * cit.0)
  (repeat i 1 6
    (sc_ndi2x1 csel cit.(- i 1) * it.i)
    (sc_nri2  cai.i cit.(- i 1) * cit.i)
  )
  (sc_ndi2x1 csel cit.6 * it.7)
  (repeat i 0 7
    (sc_xnor it.i cai.i caoB.i)
    (sc_inv2 caoB.i cao.i)
  )
)

; 8-bit latch
; lh = 0: hold current value
; lh = 1: latch value from lin to lout
(macro muxdff8 (lin lout lh clock)
  (repeat i 0 7
    (sc_dfsc clock lout.i lin.i lout.i * lh *)
  )
)

; body of circuit
; register for multiplicand: M
; inversion of control signal
(sc_inv2 c0 c0B)
(sc_inv2 c3 c3B)
(sc_inv2 c7 c7B)
;
(muxdff8 INBUS mout c1 CLK)
(comp8  mout y c6)
; connection to 8-bit adder
(adder8 GND co x y s)
; overflow and shiftin bit
(sc_dff CLK vi v *)
(sc_gpuld c0 vi *)
```

```
(sc_trib2 vi0 c0B vi * * *)
(sc_mux1 c3 c0 c3B v * vi0)
(sc_nri2  v ma7 * adl0)
(sc_ndi2x1 x.7 mout.7 * ma7)
(sc_xor mout.7 qout.0 adl1)
(sc_mux1 c7 adl1 c7B adl0 * adl)
; Accumulator: A
(tri8 s ain c3)
(dffsr8 CLK ain x adl c3 c5 c0)
(tri8 x OUTBUS c8)
; Multiplier: Q
; qdl = x.0
(tri8 INBUS qin c2)
(dffsr8 CLK qin qout x.0 c2 c5 c0)
(tri8 qout OUTBUS c9)
```

The simulation of the datapath in IRSIM is shown in appendix D.

Control Unit of Two's Complement Multiplier There are several ways to implement the control unit [5]. We adopt the *sequence counter method* to implement the control unit. The block diagram of the control unit is shown in Fig. 15. The netlist description corresponding to the control unit in Fig. 15 is shown as follows:

```
; control unit for 2's complement multiplier
; libpath declaration
(libpath "cmos/scmos")

; node declaration
(node c0 c1 c2 c3 c4 c5 c6 c7 c8 c9)
(node  CLK CLKB CLKBB0 CLKBB1 CLKBB)
(node PH1 PH2 START STARTB STOP END RESET)
(node Q1 Q2 Q3 Q4 R2)
(node qout0 COUNT7 ADD1 SHIFT1 SHIFT2 creset)

; Control Unit Design

; connect
; connect R1 c8
; connect R3 SHIFT2
; connect R4 c13

; I/O declaration
(south (in START CLK RESET qout0 COUNT7))
(north (out c0 c1 c2 c3 c5 c6 c7 c8 c9 END))
(west (in GND))

; macro declaration
; module 2 sequential counter
```

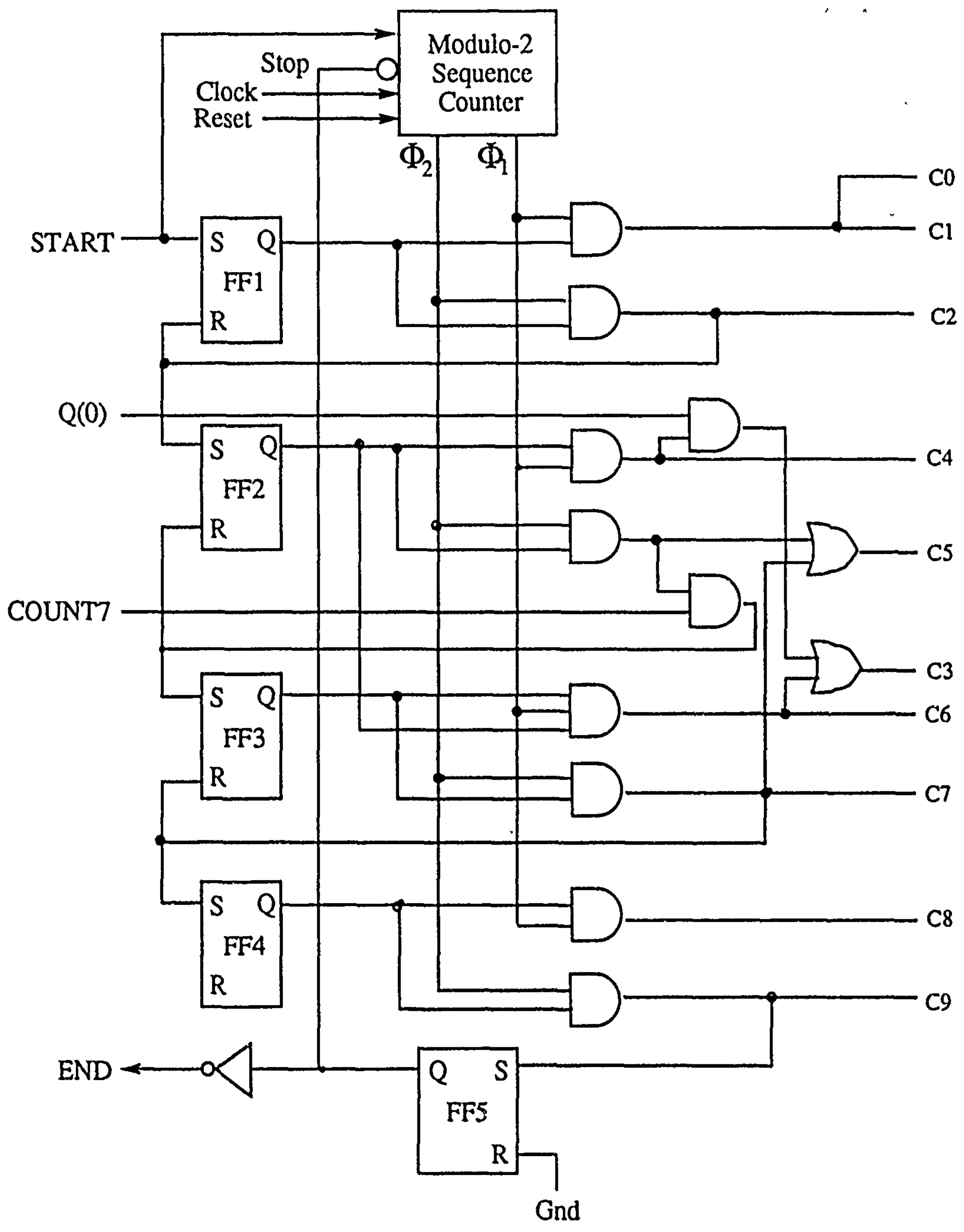

Figure 15: Block Diagram for Control Unit

```
; reset, start, stop: active low
(macro mod2 (CLOCK PH1 PH2 START STOP RESET)
  (local q ss ph01 ph02)
  (sc_ndi2x1 q ph01 * PH1)
  (sc_ndi2x1 q ph02 * PH2)
  (sc_ndi2x1 RESET STOP * ss)
  (sc_dffsr  CLOCK q  q * ss START)
  (sc_dffsr  CLOCK ph02 ph01 ph02 ss START)
)

(macro srff (CLOCK Q S R RESET)
  (local qinter resetB rr)
  (sc_inv2 RESET resetB)
  (sc_nri2 resetB R * rr)
  (sc_nrlat qinter * rr S)
  (sc_dffr CLOCK qinter Q * RESET)
)

(macro count7 (CLOCK FINISH EN RESET a b c)
  (local interclk aa bb cc bbb ccc)
  (sc_ndi2x1 CLOCK EN * interclk)
  (sc_dffr interclk aa a aa RESET)
  (sc_ndi2x1 interclk a * bbb)
  (sc_dffr bbb bb b bb RESET)
  (sc_ndi3 interclk a b * ccc *)
  (sc_dffr ccc cc c cc RESET)
  (sc_ndi3 a b c * FINISH *)
)

; body circuit
(sc_inv2 CLK CLKBB)
(sc_buff CLKBB * CLKBB0)
(sc_buff CLKBB0 * CLKBB1)
(sc_buff CLKBB1 * CLKB)
(sc_inv2 START STARTB)
(mod2 CLKBB PH1 PH2 STARTB STOP RESET)
; stage 1
(srff CLKB Q1 START c2 RESET)
(sc_ndi2x1 Q1 PH1 * c0)
(sc_buff   c0 * c1)
(sc_ndi2x1 Q1 PH2 * c2)
; R1=c2
; stage 2
(srff CLKB Q2 c2 R2 RESET)
(sc_ndi2x1 Q2 PH1 * c4)
(sc_ndi2x1 c4 qout0 * ADD1)
(sc_ndi2x1 Q2 PH2 * SHIFT1)
(sc_ndi3 Q2 COUNT7 PH2 * R2 *)
```

```
; stage 3
(srff CLKB Q3 R2 SHIFT2 RESET)
(sc_ndi3 Q3 PH1 qout0 * c6 *)
(sc_ndi2x1 Q3 PH2 * SHIFT2)
(sc_ndi2x1 COUNT7 Q3 * c7)
(sc_nri2 ADD1 c6 * c3)
(sc_nri2 SHIFT1 SHIFT2 * c5)
; R3=SHIFT2
; stage 4
(srff CLKB Q4 SHIFT2 c9 RESET)
(sc_ndi2x1 Q4 PH1 * c8)
(sc_ndi2x1 Q4 PH2 * c9)
; R4=c9
; final stage
(srff CLKB END c9 GND STARTB)
(sc_inv2 END STOP)

; counter
(sc_ndi2x1 RESET STARTB * creset)
(node h1 h2 h3 c_clk CLK0 CLK1 CLK2)
(sc_inv2 CLKB CLK0)
(sc_buff CLK0 * CLK1)
(sc_buff CLK1 * CLK2)
(sc_buff CLK2 * c_clk)
(count7 c_clk COUNT7 c4 creset h1 h2 h3)
```

The simulation of control unit in IRSIM is shown in appendix D.

Composition of the whole circuit After two modules are generated, we use FUSION SERVICE to compose these two modules and pads. Again we use NetList$^+$ to describe the composition of modules. Before running **prenet** and **prefus**, we need to create the macro cells and the leaf cells for RNL/IRSIM and FUSION respectively. As presented in early section, macro cells and leaf cells can be generated by the command **mag2rcell** and the command **mag2lcell**. Then We can write a simple NetList$^+$ description of the composition of these two blocks as follows:

```
; libpath declaration
(libpath "/nfs/si2/wuu/design/ASIC_ex/DataP/Vpnr/datapath.magic2")
(libpath "/nfs/si2/wuu/design/ASIC_ex/DataP/Vpnr")
(libpath "/nfs/si2/wuu/design/ASIC_ex/ConU/Vpnr/control.magic2")
(libpath "/nfs/si2/wuu/design/ASIC_ex/ConU/Vpnr")

; node declaration
(node INBUS OUTBUS c0 c1 c2 c3 c5 c6 c7 c8 c9 CLOCK Q0
COUNT7 END RESET START)

; IO pad declaration
(in (signals 0 7 INBUS) CLOCK RESET START)
```

```
(out (signals 0 7 OUTBUS) END)

; body of circuit
(datapath CLOCK INBUS.0 INBUS.1 INBUS.2 INBUS.3 INBUS.4 INBUS.5 INBUS.6
  INBUS.7 OUTBUS.0 OUTBUS.1 OUTBUS.2 OUTBUS.3 OUTBUS.4 OUTBUS.5 OUTBUS.6
  OUTBUS.7 c0 c1 c2 c3 c5 c6 c7 c8 c9 Q0 GND Vdd)
(control CLOCK COUNT7 END RESET START c0 c1 c2 c3 c5 c6 c7 c8 c9 Q0 GND)
```

After running **prenet**, the RNL/IRSIM netlist is generated. Then we can verify the correctness of the design before final P&R. If the circuit is verified, the FUSION specification is generated by running **prefus**. Through the same procedure used in early example, we can get the layout from FUSION SERVICE. Since these two blocks have different sizes, we generate two layouts for the final results. Each block has aspect ratio 1:1 in first result. The other result tend to make the whole layout with aspect ratio 1:1. The simulation of the whole circuit in IRSIM and the final layouts for these two cases are shown in appendix D.

Remarks We did find some design errors when we simulating the composition of datapath and control unit. We went back to find the errors from two modules. After verifying the combination of the two parts, we sent it to MOSIS' FUSION SERVICE to get the final layout.

APPENDIX

A RNL, VPNR and FUSION Result of an 1-bit Full Adder

In this appendix, we show the detailed result for 1-bit full adder.

A.1 RNL Simulation for *fa*

We have *fa*.net by running prenet. Now we can run through RNL by the following commands [9]:

```
$netlist fa.net fa.sim -tcmos-pw -u100 -fUCB
$presim fa.sim fa
```

After netlist, presim, we have *fa* ready to run RNL. For convenience' sake, we edit a file *fa*.l, which is shown as follows:

```
; The name of this control file for rnl is: full adder

; LOAD STAIDARD LIBRARY ROUTIIES
      (load "/auto/nfs/si2/uw/src/rnl/uwstd.l")
      (load "/auto/nfs/si2/uw/src/rnl/uwsim.l")

; FILE WHICH WILL LOG THE RESULTS
      (log-file "fa.rlog")

; READ II THE BIIARY IETWORK FILE
(read-network "fa")

; DEFIIE THE TIME SCALE FOR SIMULATIOI
(setq incr 100)

(setq nodes '(X Y CI SUM CO))
(chflag nodes)

(def-report '(""  "input:" X Y CI "output:" SUM CO))
```

Then we can run RNL to simulate *fa* in the following commands:

```
$rnl fa.l
Version 4.2
Loading uwsim.l
Done loading uwsim.l
; 42 nodes, transistors: enh=27 intrinsic=0 p-chan=18 dep=0 low-power=0 pullup=
0 resistor=73
< rest of simulation result is in fa.rlog >
```

The simulation process and result is stored in file *fa*.rlog, which is shown as follows:

```
; 42 nodes, transistors: enh=27 intrinsic=0 p-chan=18 dep=0 low-power=0 pullup=
0 resistor=73

l X Y CI
done
s

Step begins @ 0 ns.
CI = 0 @ 0
Y = 0 @ 0
X = 0 @ 0
CO = 0 @ .6
SUM = 0 @ 1.1

Current time= 10
input:X=0 Y=0 CI=0 output:SUM=0 CO=0
done

h X
done
s

Step begins @ 10 ns.
X = 1 @ 0
SUM = 1 @ 1.1

Current time= 20
input:X=1 Y=0 CI=0 output:SUM=1 CO=0
done

l X
done
h Y
done
s

Step begins @ 20 ns.
Y = 1 @ 0
X = 0 @ 0

Current time= 30
input:X=0 Y=1 CI=0 output:SUM=1 CO=0
done

h X
done
s

Step begins @ 30 ns.
X = 1 @ 0
CO = 1 @ .6
SUM = 0 @ 1.1

Current time= 40
input:X=1 Y=1 CI=0 output:SUM=0 CO=1
done

l X Y
done
h CI
```

```
done
s

Step begins @ 40 ns.
CI = 1 @ 0
Y = 0 @ 0
X = 0 @ 0
CO = 0 @ 1
SUM = 1 @ 1.5

Current time= 50
input:X=0 Y=0 CI=1 output:SUM=1 CO=0
done

h X
done
s

Step begins @ 50 ns.
X = 1 @ 0
CO = 1 @ .9
SUM = 0 @ 1.4

Current time= 60
input:X=1 Y=0 CI=1 output:SUM=0 CO=1
done

l X
done
h Y
done
s

Step begins @ 60 ns.
Y = 1 @ 0
X = 0 @ 0

Current time= 70
input:X=0 Y=1 CI=1 output:SUM=0 CO=1
done

h X
done
s

Step begins @ 70 ns.
X = 1 @ 0
SUM = 1 @ .8

Current time= 80
input:X=1 Y=1 CI=1 output:SUM=1 CO=1
done
(exit)
```

A.2 VPNR Standard Cell P&R for *fa*

We obtain v_*fa*.net after running prenet in section 5. Now we run VPNR [6] by using the command:

```
$vpnr v_fa.net
```

And the resulting layout is shown in Fig. 16

A.3 FUSION SERVICE for *fa*

We obtain *fa*.spc and *fa*.lib after running **prefus** in section 5. Now we feed *fa*.spc and *fa*.lib through **lif2fus** to get *fa*.fu, that is,

```
$lif2fus
file the contains the names of your lif files? fa.lib
file the contains your fusion specifications? fa.spc
creating fa.fu
creating fa.cf
...Doing Fuse...
...Doing Compose...
4.7u 4.3s 1:41 9% 4+5k 142+58io 0pf+0w
```

Then you will see *fa*.fu and *fa*.fu are generated. You can send *fa*.fu to MOSIS SERVICE, and you will get *fa*.fused back. Then you can get *fa*.cif by running **fus2cif**, that is,

```
$fus2cif
Remember to make sure that there is no calls.cif and wiring.cif
in this directory. Else they might be overwritten!
filename of your library CIF? fa.cf
filename of your fusion result? fa.fused
filename of your output? (xxx.cif) fa.cif
2.5u 2.5s 1:07 7% 3+3k 79+25io 0pf+0w
```

After getting *fa*.cif, we can used **cif read** in *magic* to look at the result. The Layout of the chip is shown in Fig. 17, and the internal layout of *fa* is shown in Fig. 18.

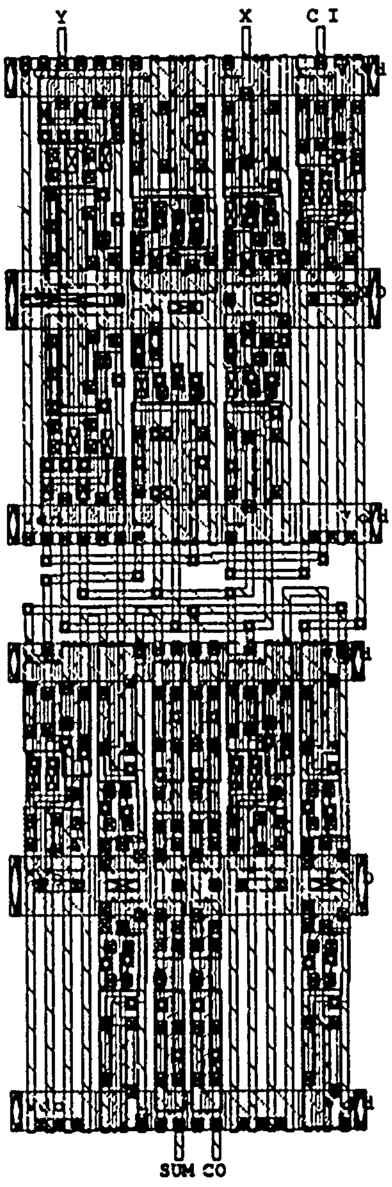

Figure 16: Layout of *fa* from VPNR

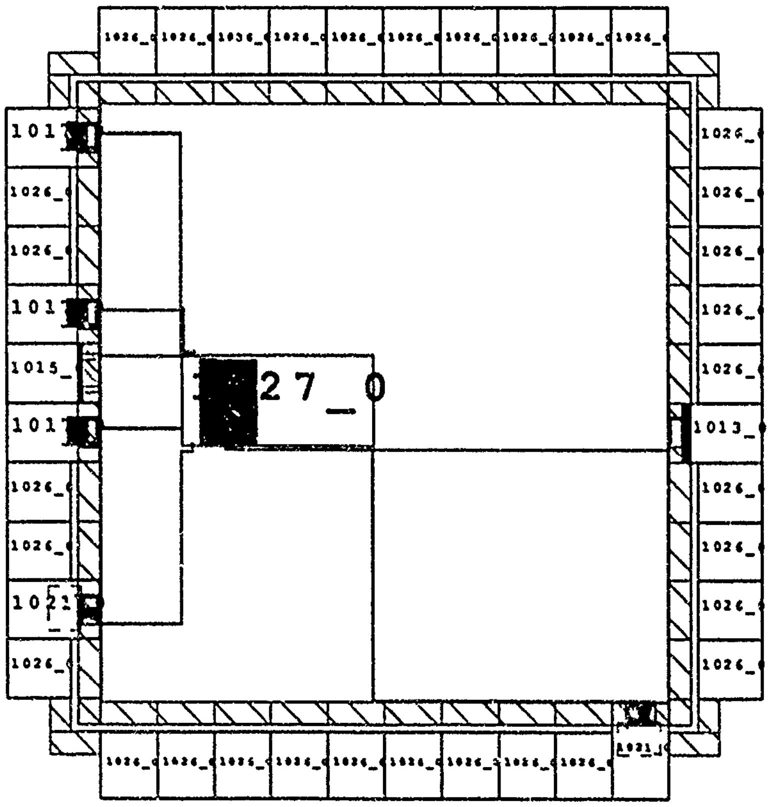

Figure 17: Chip Layout of *fa* from FUSION

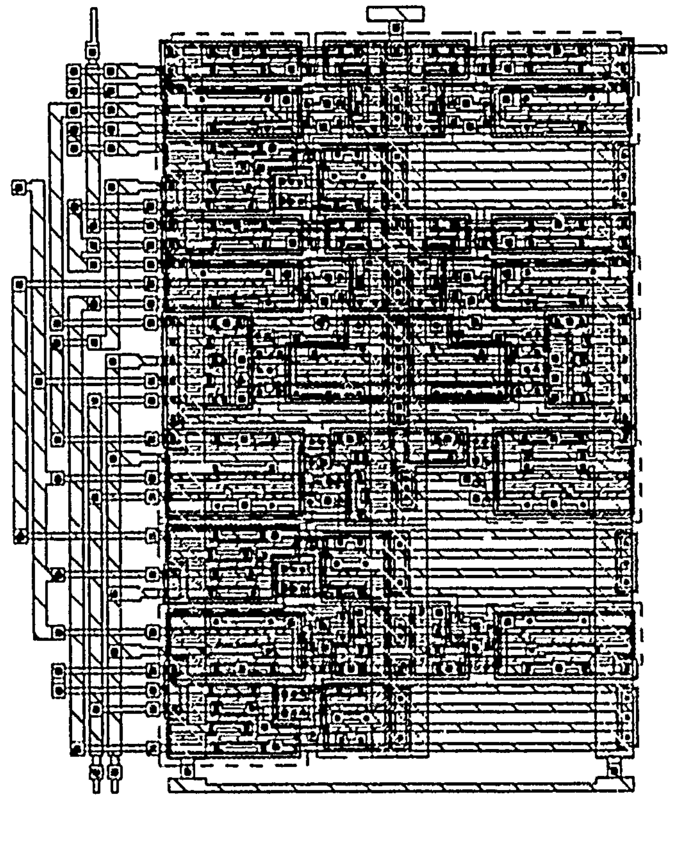

Figure 18: Internal Layout of *fa* from FUSION

B RNL, VPNR and FUSION Result of 4-bit Parallel Multiplier

In this appendix, we show the detailed result for 4-bit multiplier. The design done by using the *connect* construct of NetList$^+$ is named *pm0*. The design done without using the *connect* is named *pm1*. The design done with floorplan specified by user is named *pm2*.

B.1 RNL Simulation for *pm0*

We obtain *pm0*.net by running prenet. Now we can run through RNL by following command [9],

```
$netlist pm0.net pm0.sim -tcmos-pw -u100 -fUCB
$presim pm0.sim pm0
```

After netlist, presim, we have *pm0* ready to run RNL. For convenience' sake, we edit a file *pm0*.l, which is shown as follow,

```
; The name of this control file for rnl is: 4-bit multiplier

; LOAD STANDARD LIBRARY ROUTINES
      (load "/auto/nfs/si2/uw/src/rnl/uwstd.l")
      (load "/auto/nfs/si2/uw/src/rnl/uwsim.l")

; FILE WHICH WILL LOG THE RESULTS
      (log-file "pm0.rlog")

; READ IN THE BINARY NETWORK FILE
      (read-network "pm0")

; DEFINE THE TIME SCALE FOR SIMULATION
      (setq incr 100)

(defvec '(bin Xin X.4 X.3 X.2 X.1))
(defvec '(bin Yin Y.4 Y.3 Y.2 Y.1))
(defvec '(bin Pout P.8 P.7 P.6 P.5 P.4 P.3 P.2 P.1))

(def-report '( "State of input:"
              (vec Xin) "\n"
              (vec Yin) "\n"
              (vec Pout)))

(setq NN 4)

(setq nodes nil)
(repeat i 1 (* 2 NN)
  (setq nodes (cons P.(eval i) nodes))
)

(chflag nodes)

(repeat i 1 NN
  (l '(X.(eval i)))
  (l '(Y.(eval i)))
)
(s nil)

(h '(Y.1))
(repeat i 1 ##
  (s nil)
  (h '(X.(eval i)))
)

(repeat i 2 ##
  (s nil)
  (h '(Y.(eval i)))
)
(s nil)

(repeat i 1 ##
  (l '(X.(eval i)))
  (l '(Y.(eval i)))
)
(s nil)

(repeat i 1 ##
  (h '(X.(eval i)))
  (h '(Y.(eval i)))
)
(s nil)
```

Then we can run RNL to simulate *pm0* in following command

```
$rnl pm0.l
Version 4.2
Loading uwsim.l
Done loading uwsim.l
; 42 nodes, transistors. enh=27 intrinsic=0 p-chan=18 dep=0 low-power=0 pullup =0 resistor=73
< rest of simulation result is in pm0.rlog >
```

The simulation process and result is stored in file *pm0*.rlog, which is shown as follow,

```
; 359 nodes, transistors. enh=272 intrinsic=0 p-chan=240 dep=0 low-power=0 pullup=0 resistor=492

Step begins @ 0 ns.
p.0.2 = 0 @ 1.5
p.0.3 = 0 @ 2.8
p.0.4 = 0 @ 4.1
p.0.5 = 0 @ 5.4
P.8 = 0 @ 5.8
P.5 = 0 @ 6.2
P.7 = 0 @ 6.5
P.6 = 0 @ 6.5
State of input:
Current time= 10
Xin=0b0000
Yin=0b0000
Pout=0b00000000

Step begins @ 10 ns.
State of input:
Current time= 20
```

```
Xin=0b0000
Yin=0b0001
Pout=0b00000000

Step begins @ 20 ns.
p.0.2 = 1 @ 1.6
State of input:
Current time= 30
Xin=0b0001
Yin=0b0001
Pout=0b00000001

Step begins @ 30 ns.
p.0.3 = 1 @ 2.4
State of input:
Current time= 40
Xin=0b0011
Yin=0b0001
Pout=0b00000011

Step begins @ 40 ns.
p.0.4 = 1 @ 3.2
State of input:
Current time= 50
Xin=0b0111
Yin=0b0001
Pout=0b00000111

Step begins @ 50 ns.
p.0.5 = 1 @ 4
State of input:
Current time= 60
Xin=0b1111
Yin=0b0001
Pout=0b00001111

Step begins @ 60 ns.
p.0.3 = 0 @ 2.4
p.0.4 = 0 @ 2.9
p.0.4 = 1 @ 3.9
P.7 = 1 @ 5.1
P.6 = 1 @ 6.8
P.7 = 0 @ 7.2
State of input:
Current time= 70
Xin=0b1111
Yin=0b0011
Pout=0b00101101

Step begins @ 70 ns.
p.0.4 = 0 @ 2.4
p.0.5 = 0 @ 2.9
P.7 = 1 @ 3.4
p.0.5 = 1 @ 3.9
P.6 = 0 @ 4.8
P.6 = 1 @ 5.6
P.7 = 0 @ 5.8
P.8 = 1 @ 6
P.7 = 1 @ 6.5
P.8 = 0 @ 6.9
```

```
State of input:
Current time= 80
Xin=0b1111
Yin=0b0111
Pout=0b01101001

Step begins @ 80 ns.
P.5 = 1 @ 2.1
p.0.5 = 0 @ 2.4
P.8 = 1 @ 3.1
P.5 = 0 @ 3.8
State of input:
Current time= 90
Xin=0b1111
Yin=0b1111
Pout=0b11100001

Step begins @ 90 ns.
p.0.2 = 0 @ 1.2
P.7 = 0 @ 3.3
P.6 = 0 @ 3.8
P.8 = 0 @ 4.7
State of input:
Current time= 100
Xin=0b0000
Yin=0b0000
Pout=0b00000000

Step begins @ 100 ns.
p.0.2 = 1 @ 1.4
p.0.5 = 1 @ 1.6
p.0.4 = 1 @ 1.6
p.0.3 = 1 @ 1.6
P.7 = 1 @ 2.4
P.6 = 1 @ 2.6
P.5 = 1 @ 2.6
p.0.3 = 0 @ 2.9
P.8 = 1 @ 4
P.5 = 0 @ 4
p.0.4 = 0 @ 4.4
p.0.5 = 0 @ 5.9
State of input:
Current time= 110
Xin=0b1111
Yin=0b1111
Pout=0b11100001
(exit)
```

B.2 VPNR Standard Cell P&R for *pm0*

We have v_*pm0*.net after running prenet in section 5. Now we can run through VPNR [6] by using command,

$vpnr v_*pm0*.net

And the layout is shown in Fig. 19

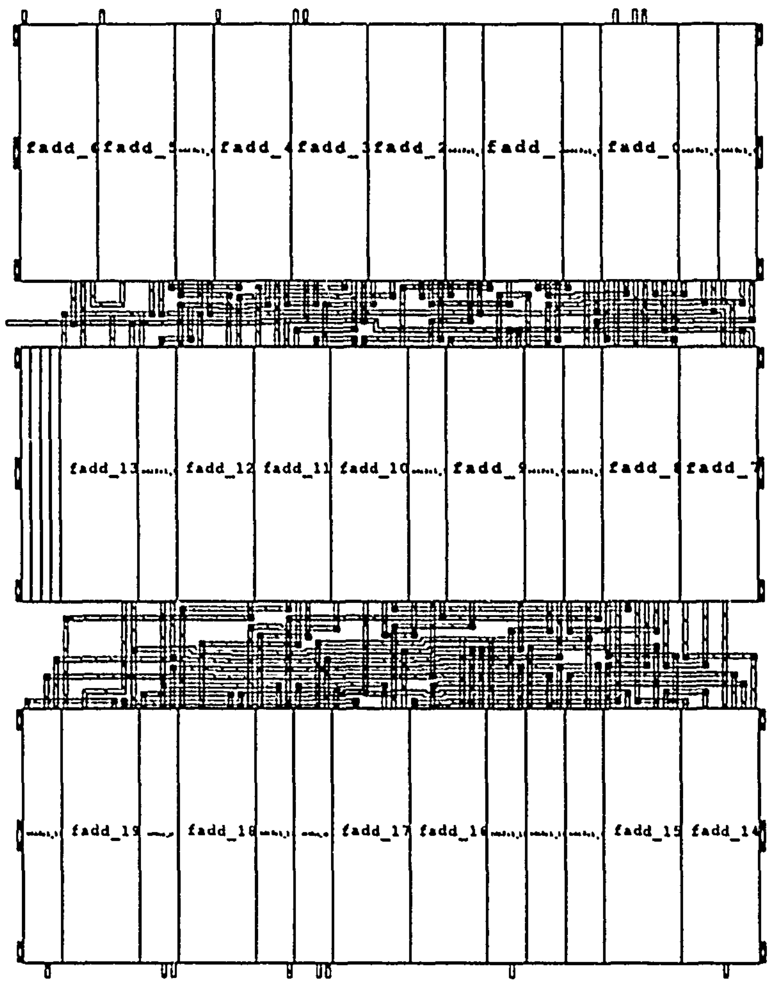

Figure 19: Layout of *pm0* from VPNR

B.3 FUSION SERVICE for *pm0*

We obtain *pm0*.spc and *pm0*.lib after running **prefus** in section 5. Now we feed *pm0*.spc and *pm0*.lib through lif2fus to get *pm0*.fu, that is,

```
$lif2fus
file the contains the names of your lif files? pm0.lib
file the conatins your fusion specifications? pm0.spc
creating pm0.fu
creating pm0.cf
...Doing Fuse...
...Doing Compose...
4.7u 4.3s 1:41 9% 4+5k 142+58io 0pf+0w
```

Then you will see *pm0*.fu and *pm0*.fu are generated. You can send *pm0*.fu to MOSIS SERVICE, and you will get *pm0*.fused back. Then you can get *pm0*.cif by running **fus2cif**, that is,

```
$fus2cif
Remember to make sure that there is no calls.cif and wiring.cif
in this directory. Else they might be overwritten!
filename of your library CIF? pm0.cf
filename of your fusion result? pm0.fused
filename of your output? (xxx.cif) pm0.cif
2.5u 2.5s 1:07 7% 3+3k 79+25io 0pf+0w
```

After getting *pm0*.cif, we can used **cif read** in *magic* to look at the result. The layout of the chip is shown in Fig. 20.

B.4 FUSION SERVICE for *pm1* and *pm2*

We run through the same procedure, as described in previous sections for *pm0*, to prepare *pm1* and *pm2* for FUSION jobs. After MOSIS' FUSION SERVICE, we get *pm1*.cif and *pm2*.cif for the layout without *connect* and the layout with floorplan respectively. The Layouts of the chips are shown in Fig. 21 and Fig. 22.

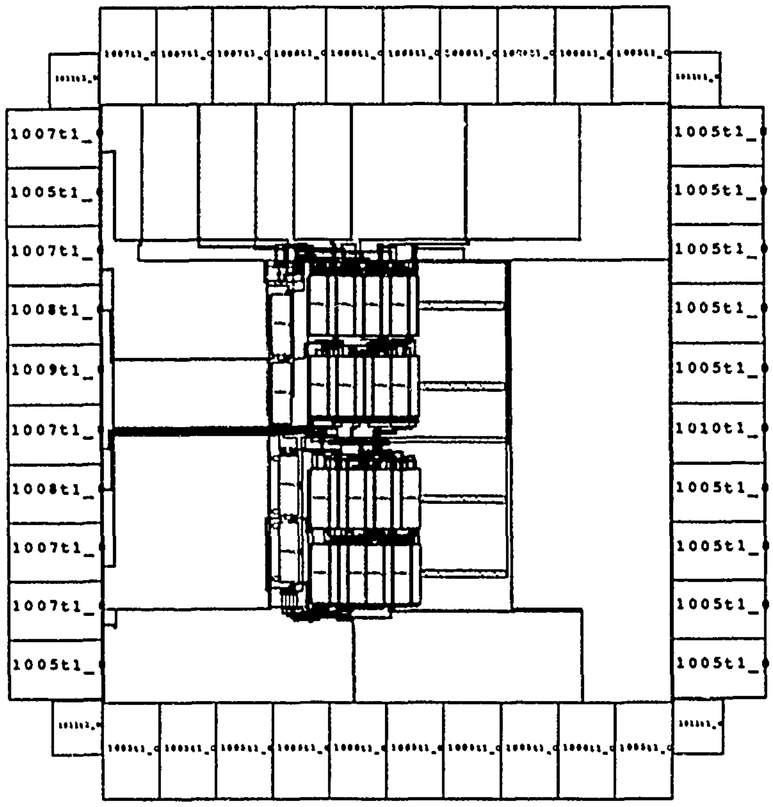

Figure 20: Chip Layout of *pm0* from FUSION

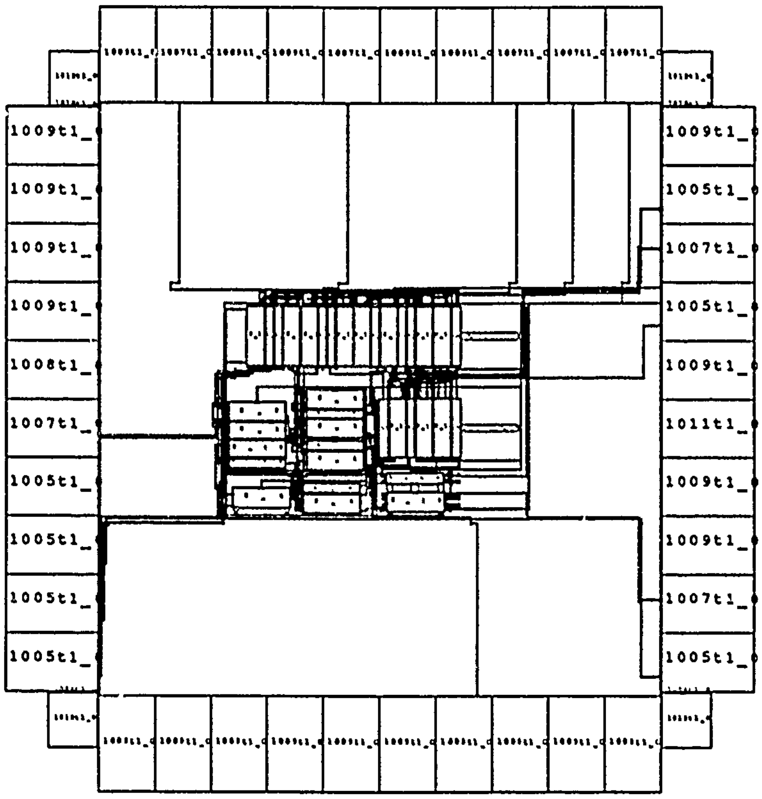

Figure 21: Chip Layout of *pm1* from FUSION

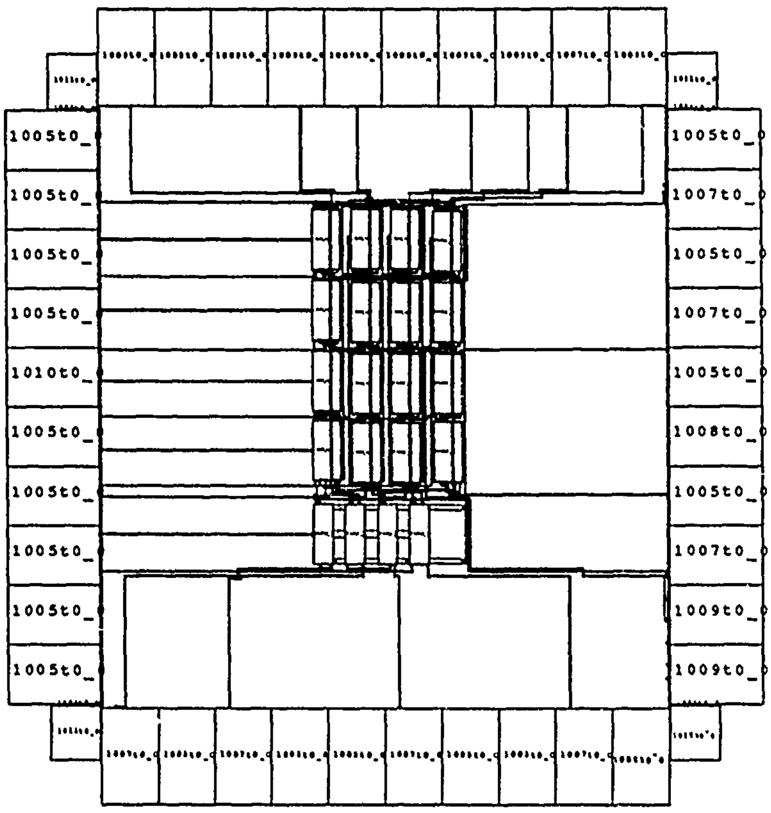

Figure 22: Chip Layout of *pm2* from FUSION

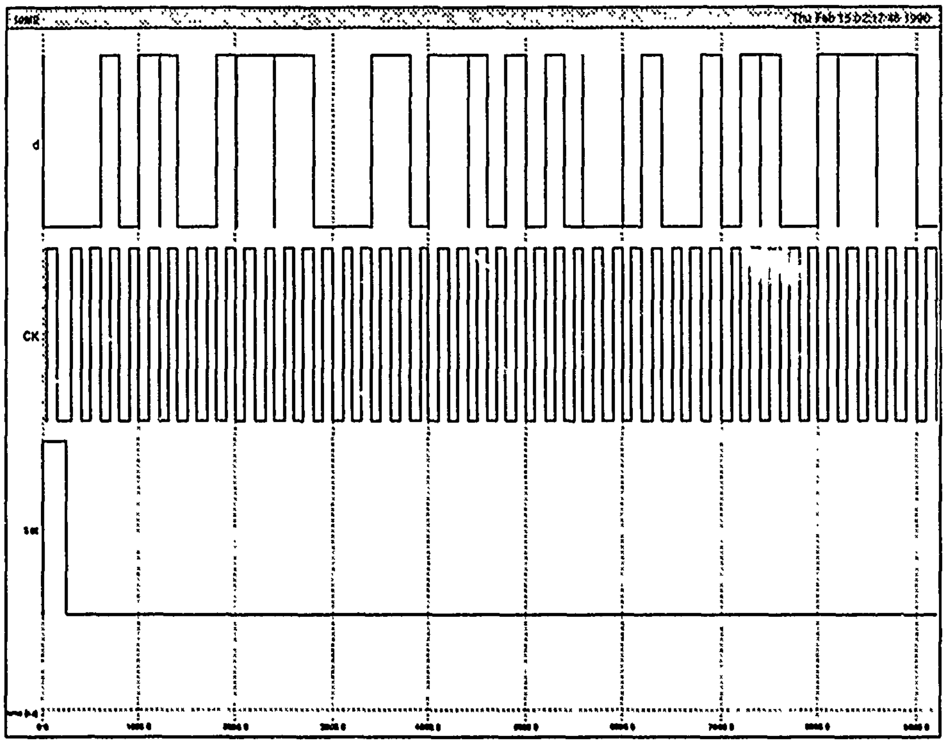

Figure 23: Simulation Result of *prg* from IRSIM

C IRSIM, VPNR Result of a Pseudorandom Number Generator

In this appendix, we show the detailed result for 5-bit pseudorandom number generator.

C.1 IRSIM Simulation for *prg*

We have *prg*.net by running **prenet**. We only need to run through **netlist** to get .sim file for IRSIM.

$netlist *prg*.net *prg*.sim

Now we have *prg*.sim ready to run IRSIM. The simulation result can either be viewed on an X window system or be outputed in POSTSCRIPT format, which is fed into any POSTSCRIPT printer as shown in Fig. 23.

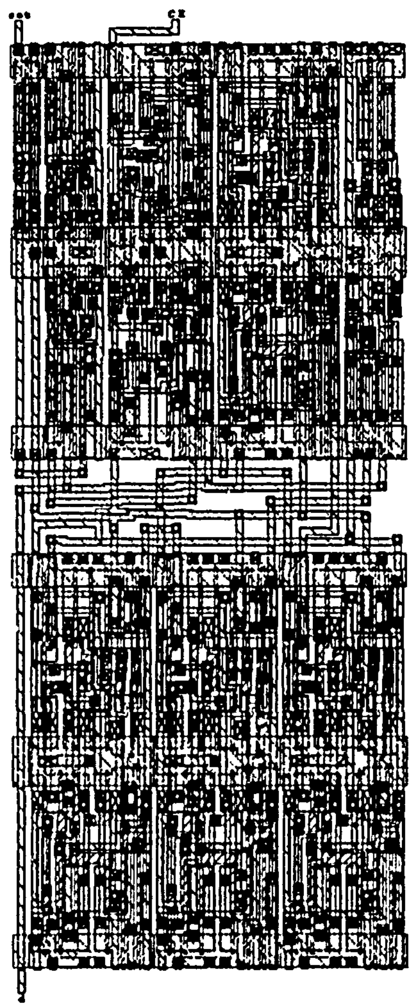

Figure 24: Layout of *prg* from VPNR

C.2 VPNR Standard Cell P&R for *prg*

We have v_*prg*.net after running **prenet** in section 5. Now we can run through VPNR [6] by using the command,

$vpnr v_*prg*.net

And the layout is shown in Fig. 24

D Simulation Result and Layout of 4-bit Two's Complement Multiplier

In this appendix, we show the detailed simulation result and layout of the 4-bit two's complement multiplier.

D.1 IRSIM Simulation for the Datapath and the Control Unit of *4-bit Two's Complement Multiplier*

We obtained *datapath*.net by running **prenet**. We only need to run through **netlist** to get the .sim file for IRSIM.

```
$prenet datapath
$netlist datapath.net datapath.sim
```

Fig. 25 illustrates the IRSIM result of running *datapath*.sim. Similarly, we have the IRSIM result for control unit shown in Fig. 26.

D.2 IRSIM Simulation for the Whole Design

There are two modules generated seperately. Before the final P&R, we simulated the composition of them. We didn't simulate exhaustively, but picked four typical cases to test. They are show in Fig. 27, 28, 29, and 30.

D.3 FUSION SERVICE for *4-bit Two's Complement Multiplier*

There are two layouts generated for the design in terms of different shapes of the datapath and the control unit. In first result, the layout of the datapath and the layout of control unit have aspect ratio 1:1, and the FUSION result is shown in Fig. 31. In this layout, the bigger block is the datapath, and the smaller one is the control unit. In second result, we want the final layout with aspect ratio 1:1. We assign the aspect ratio while blocks are generated. And the FUSION result is shown in Fig. 32. The second layout is 9.2% less than the first one.

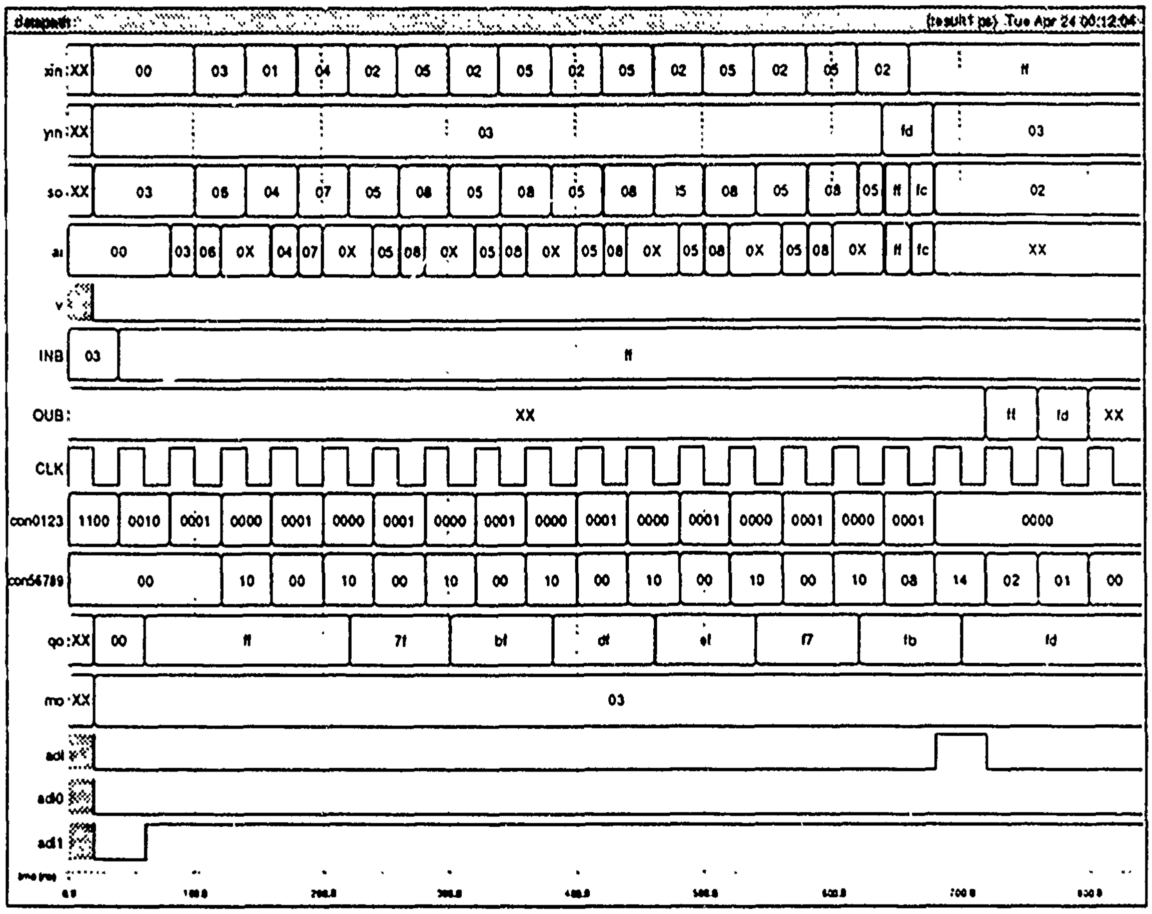

Figure 25: Simulation of Datapath

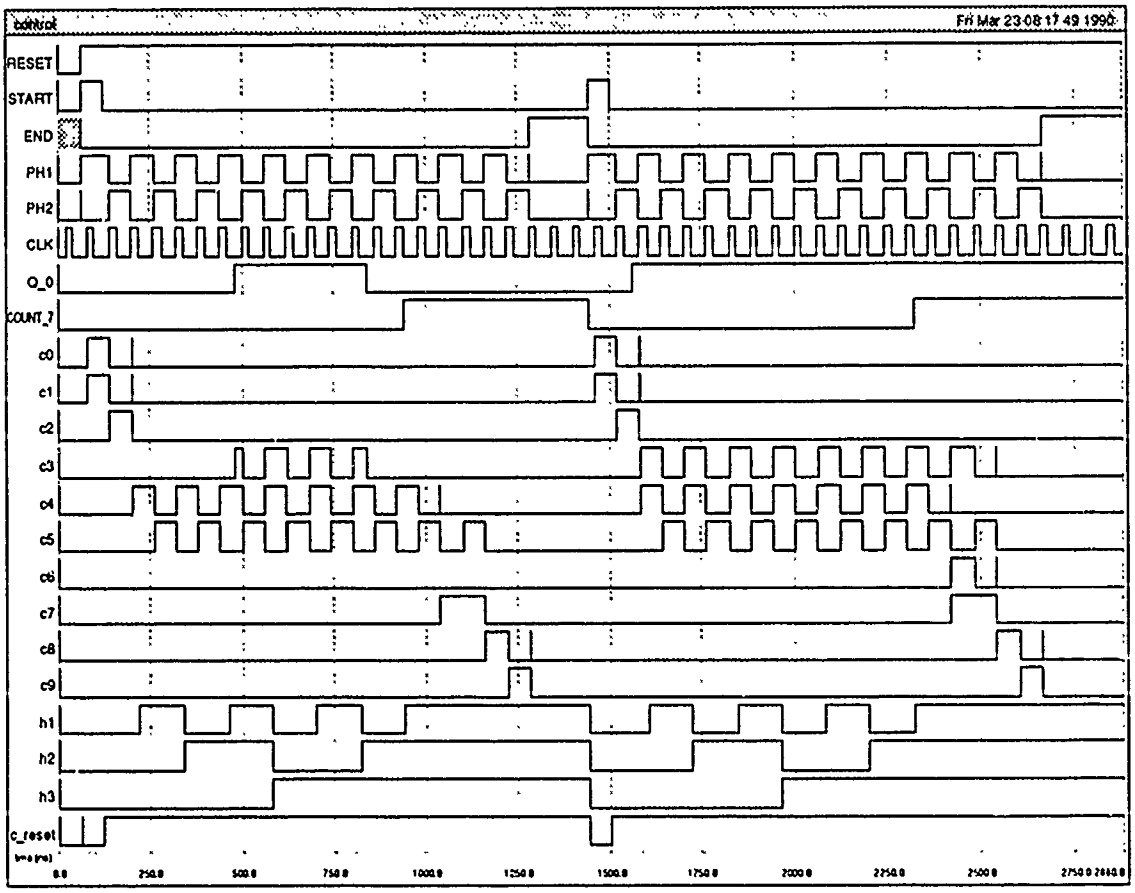

Figure 26: Simulation of Control Unit

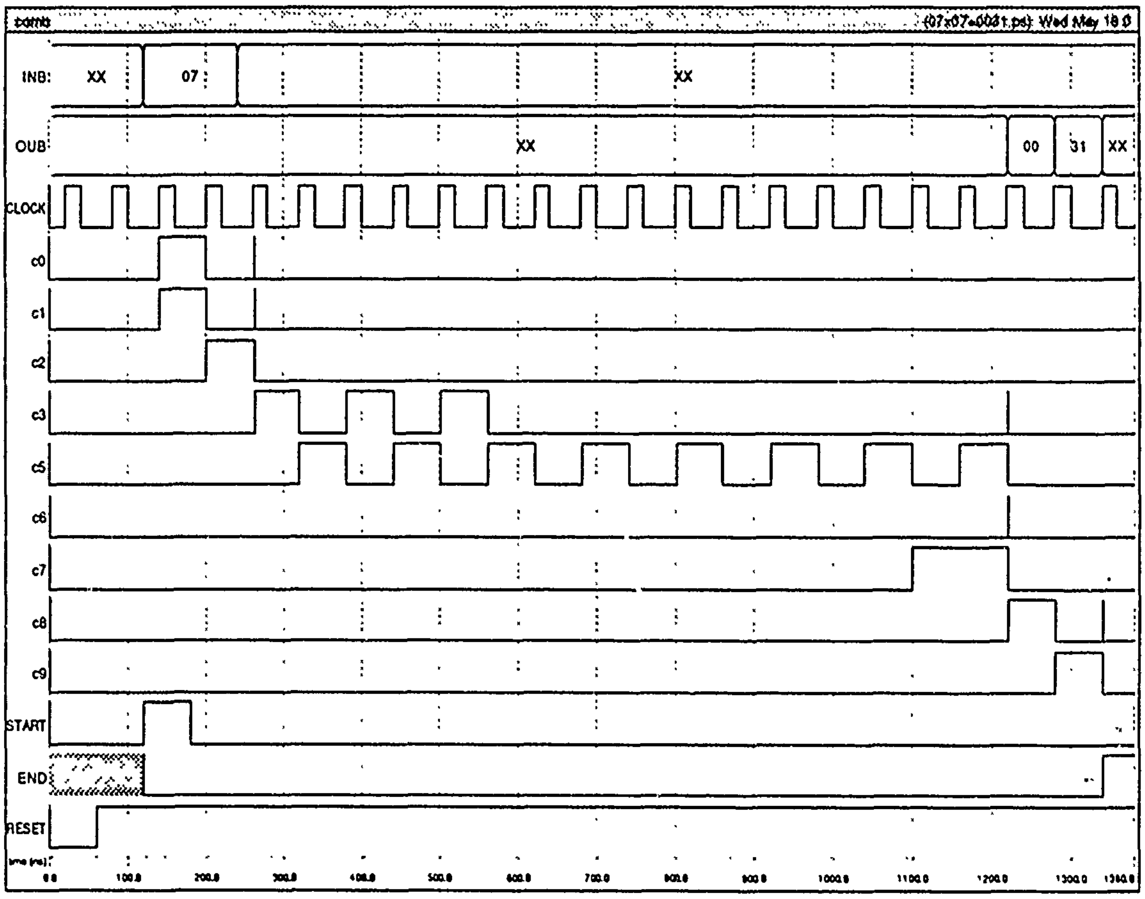

Figure 27: Test 1: $07 \times 07 = 0031$

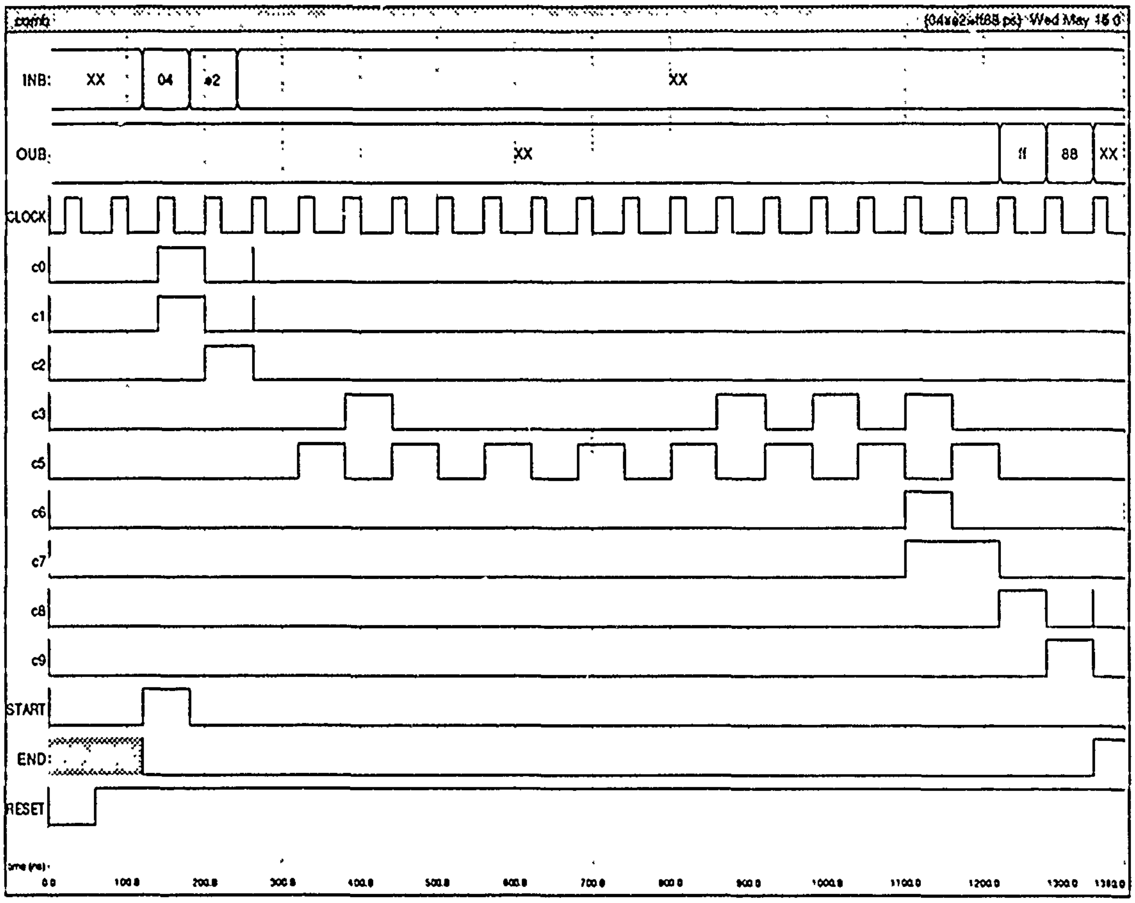

Figure 28: Test 2: $04 \times e2 = ff88$

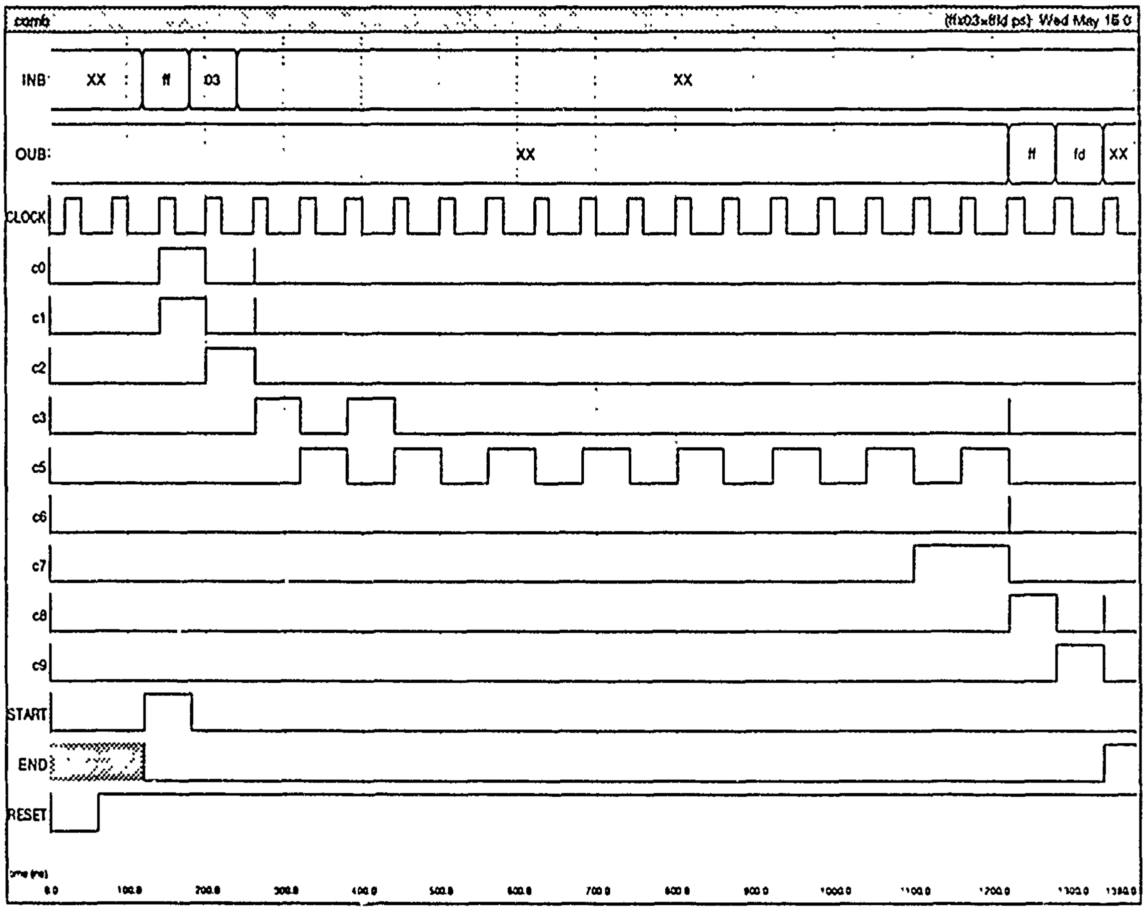

Figure 29: Test 3: $ff \times 03 = fffd$

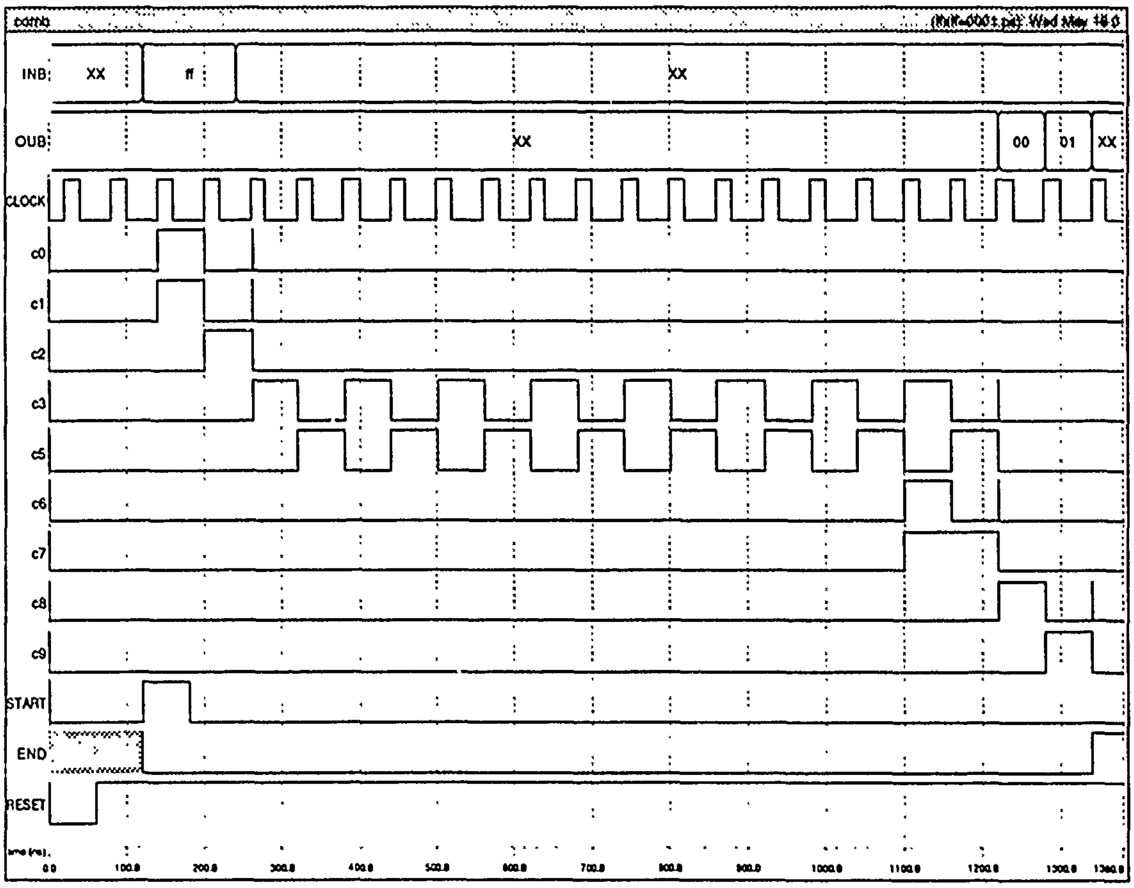

Figure 30: Test 4: $ff \times ff = 0001$

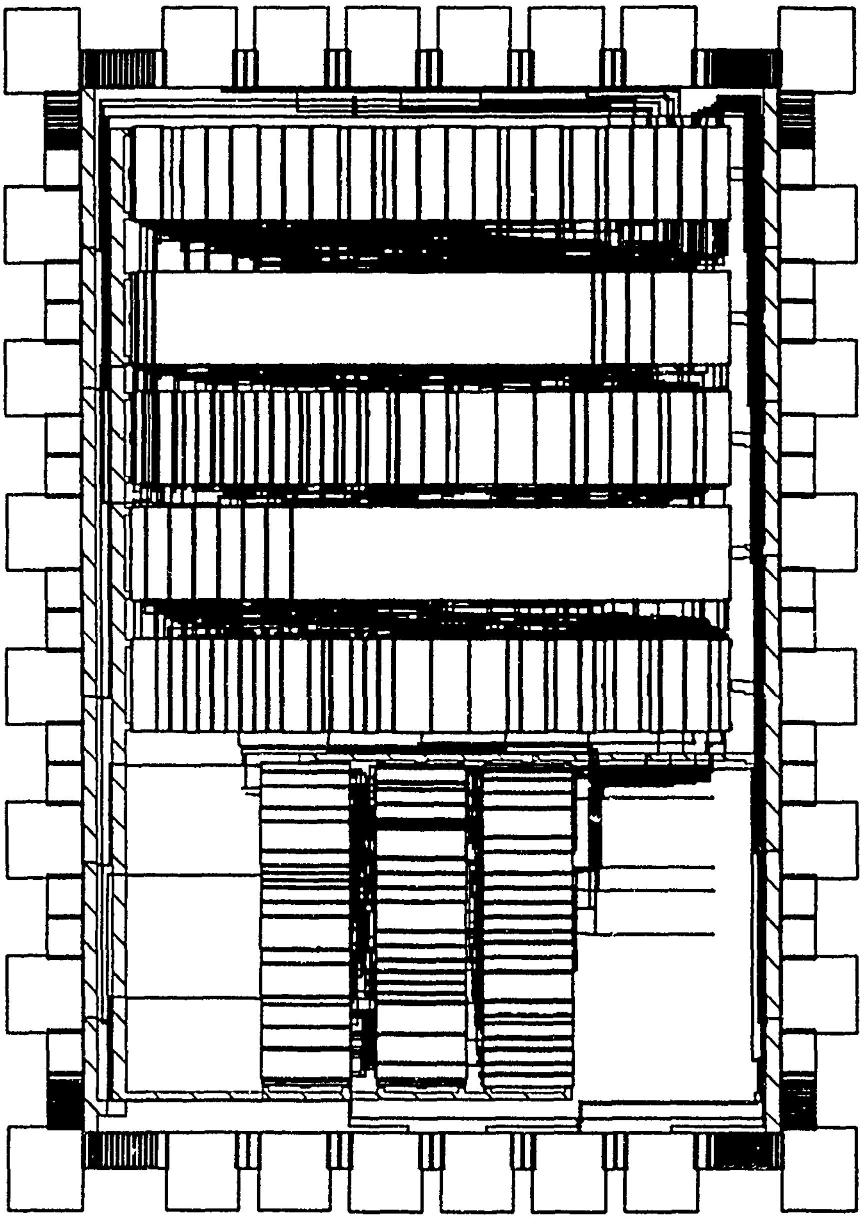

Figure 31: Two's Complement Multiplier without Adjustment

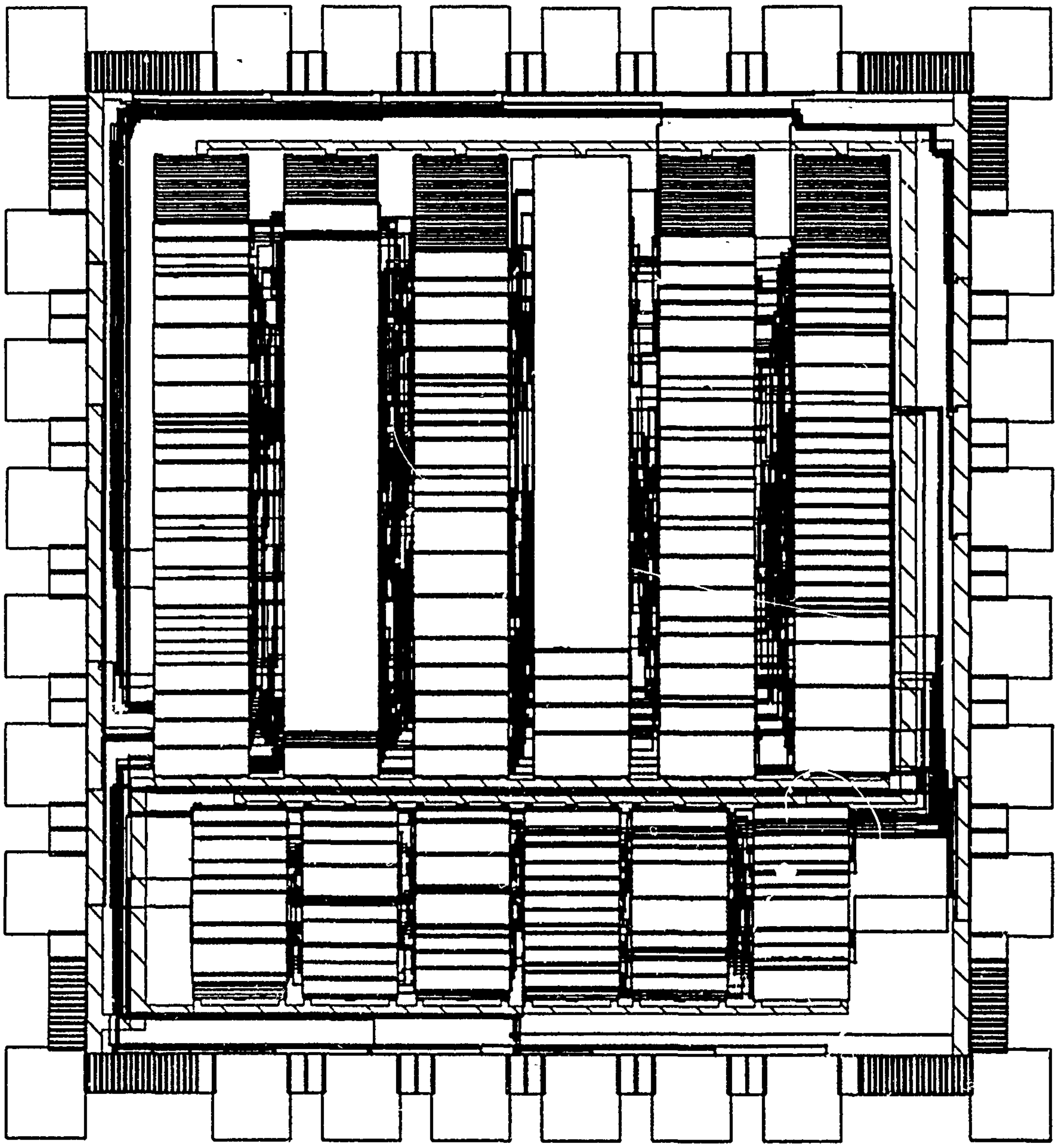

Figure 32: Two's Complement Multiplier with Adjustment

E Functions and Macros of NetList$^+$

In this appendix, we summarize the functions and macros which are available in NetList$^+$.

(libpath "*dir*") declares the directory of the cell library. There can be more than one libraries in one circuit.

(load "*filename*") insert other NetList$^+$ files to the current NetList$^+$.

(node *n1 n2* ...) declares nodes *n1 n2* ... etc. The nodes has to be declared before any other statements except **libpath**.

(in *i_n1 i_n2* ...) declares input pads *i_n1 i_n2* ... etc. Each i_pad can be either a node *in0* or a pair of nodes, (*in1 in2*), where *in0 in1 in2* should be declared in the **node** statement. Each i_pad can also be arrays of input pads with the form
(signals *low high i_n1 i_n2* ...)
low and *high* are non-negative integer. i_pad can be a simple pad or nested array.

(out *o_n1 o_n2* ...) declares output pads *o_n1 o_n2* ... etc. Each o_pad can be either a single node declared in the **node** statement. Each o_pad can also be a (nested) arrays of output pads like the form in i_pad.

(io *io_n1 io_n2* ...) declared io pads *io_n1 io_n2* ... etc. Each io_pad can be either a three-tuple or a four-tuple of nodes which are declared in the **node** statement. The nodes in three-tuple are in_node, out_node, enable_node in order. The nodes in four-tuple are in_node, in_bar_node, out_node, enable_node in order. Also each io_pad can be a (nested) arrays of io pads with the same form as i_pad.

(*cellname s_n1 s_n2* ...) is a cell with name "*cellname*", and it has the nets *s_n1 s_n2* ... connected to other cells. The cell can be library cells, user-designed cells or macro cells which are declared before refered. Each *cellname* has fixed number of ports in order. Each port has to connect to an appropriate net (or node). If some ports are floating, "*" is required to put in corresponding port of the cell.

(repeat *index low high somestuff*) specifies iteration of *somestuff*, where integer variable *index* will be given successive values from *low* to *high*. *somestuff* can be cells, repeats of *another_somestuff*, or floorplan of *another_somestuff*.

(*direction somestuff*) specifies floorplan of *somestuff*. *direction* can be either of **left**, **right**, **up**, or **down**. They are used to specify the relative position of cells in *somestuff*. **right** means to order cell from left to right, and there are similar meanings for **left**, **up**, and **down**. *somestuff* can be cells, repeats of *another_somestuff*, or floorplan of *another_somestuff*.

(macro *cellname* (*p_n1 p_n2* ...) (local *l_n1 l_n2* ...) *somestuff*) is macro declaration with cell name "*cellname*", and it has formal parameters *p_n1 p_n2* ... etc and local nodes *l_n1 l_n2* ... etc. Parameter *p_ni* can be either a simple node or interger type variable in form

(**int** *v_name*)

or array of signals in the form of

(**signals** *nodes* or *nested signals*).

local declaration just same as **node** except nodes declared in **local** can be used inside **macro** only. *somestuff* is descriptions of cells, repeats of cells, or floorplan of cells.

(setq *name constant*) set the *name* to the value constant. setq can only use grobally.

(connect *n1 n2* ...) *n1 n2* ... are electrically connected. This function can be used for RNL and VPNR. So far we have not implemented in FUSION.

F Pad Library at FUSION

In this appendix, we will list the pad libraries available in FUSION service. In the table, *xx*PC means *xx*-pin pad frame with grass on the cornors. *xx*P means *xx*-pin pad frame without grass on the cornors. In pad format, -i1 means there is one input signal, *in*, from a input pad, and -i2 means there are two input signals, *in* and *in_b*, from a input pad, and -i3 means there are three input signals, *in* and *in_b* and *in_un*, from a input pad. The relation among signals *in*, *in_b*, *in_un* are shown in Fig. 33.

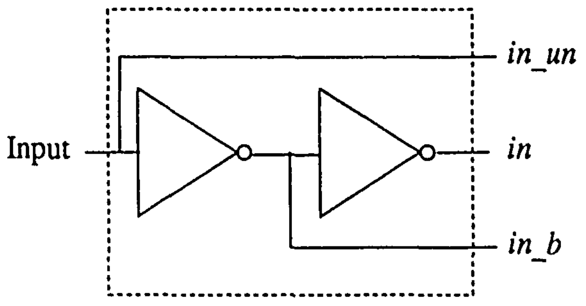

Figure 33: Input Pad

F.1 Pads for SCMOS

SCMOS 2 Micron Pads:

PAD_LIBRARY	PAD_FRAME	PAD FORMAT
MOSIS_SCN2U_PADS	28PC;40PC;64PC;84PC;132PC	-i2
TINY_SCN2U_PADS	28PC;40PC;64PC;84PC;132PC	-i2
SLLU2_PADS	28PC;40PC;64PC;84PC;132PC	-i2
TINY_SLLU2_PADS	28PC;40PC;64PC;84PC;132PC	-i2
SCALABLE_MIT_PADS	28P, 40P, 64P, 84P, 132P	-i1

In pad library SLLU2_PADS, there is a special pad frame 40PC22X22, which is called *tiny chip* with special rate at MOSIS SERVICE.

SCMOS 1.6 Micron Pads:

PAD_LIBRARY	PAD_FRAME	PAD FORMAT
MOSIS_SCN16U_PADS	28P, 40P, 64P, 84P, 132P	-i1
TINY_SCN16U_PADS	28PC,40PC,64PC,84PC,132PC	-i3
SLLU16_PADS	28P, 40P, 64P, 84P, 132P	-i1
TINY_SLLU16_PADS	28PC,40PC,64PC,84PC,132PC	-i3

References

[1] R. Ayres. FUSION: A New MOSIS Service. Technical Report ISI/TM-87-194, Information Science Institute, University of Southern California, October 1987.

[2] R. Ayres. "Completely Automatic Completion of VLSI Designs". *IEEE Transactions on Computer-Aided Design*, CAD-9(2):194–202, 1990.

[3] Chorng-Yeong Chu. "Improved Models for Switch-Level Simulation". Technical Report CSL-TR-88-368, Stanford University, Nov. 1988.

[4] G. Hamachi, R. Mayo, J. Ousterhout, W. Scott, G. Taylor, et al. Berkeley CAD Tools User's Manual. Technical report, Computer Science Division, EECS Department, University of California at Berkeley, 1986.

[5] J. P. Hayes. *Computer Architecture and Organization.* McGraw, 1978.

[6] MCNC. VPNR: Vanilla Place aNd Route System. Technical report, Microelectronics Center of North Carolina, 1988.

[7] Carver Mead and Lynn Conway. *Introduction to VLSI Systems.* Addison Wesley, MA, 1980.

[8] L. W. Nagel. SPICE2: A Computer Program to Simulate Semiconductor Circuits. Technical Report Memo ERL-M520, Computer Science Division, EECS Department, University of California at Berkeley, May 1975.

[9] NW LIS. VLSI Design Tools Reference Manual, Release 3.2. Technical Report TR#88-09-01, Northwest Laboratory for Integrated Systems, University of Washington, September 1988.

[10] A. Salz and M. Horowitz. "IRSIM: An Incremental MOS Switch-Level Simulator". In *Proceedings of the 26th Design Automation Conference*, pages 173–178, 1989.

Manual Pages for Commands

All Synposes for tools we support are listed as follow,

fus2cif generate .cif file from .fused file

lif2fus generate .fu file for FUSION SERVICE

mag2lcell generate .lif cell for fusion_spc from .mag file

mag2lif generate .lif cell for fusion_spc from .mag file

mag2rcell generate macro cell for RNL and VPNR netlist from .mag file

prefus generate fusion_spc and fusion library from netlist$^+$

prenet generate RNL and VPNR netlist from netlist$^+$

sim2cell generate RNL and VPNR macro cell .net from .sim file

NAME

fus2cif – generate *.cif* file from from *.fused* file

SYNOPSIS

fus2cif

DESCRIPTION

Fus2cif is an interactive program, and it reads *name*.fused and *name*.cf to generate *name*.cif.

EXAMPLE

```
$fus2cif
Remember to make sure that there is no calls.cif and wiring.cif
in this directory. Else they might be overwritten!
filename of your library CIF? project.cf
filename of your fusion result ? project.fused
filename of your output? (xxx.cif) project.cif
2.5u 2.5s 1:07 7% 3+3k 79+25io 0pf+0w
```

SEE ALSO

FUSION: A New MOSIS Service

AUTHOR

Shih-Lien Lu (USC/ISI)

NAME

lif2fus – generate *.fu* file for FUSION SERVICE

SYNOPSIS

lif2fus

DESCRIPTION

Lif2fus is an interactive program, and it reads *name*.lib and *name*.spc to generate *name*.cf and *name*.fu.

EXAMPLE

```
$lif2fus
file the contains the names of your lif files? project.lib
file the conatins your fusion specifications? project.spc
output file name? (without extension)project
creating project.fu
creating project.cf
```

SEE ALSO

FUSION: A New MOSIS Service

AUTHOR

Shih-Lien Lu (USC/ISI)

NAME

mag2lcell – generate *.lif* cell for fusion_spc from *.mag* file

SYNOPSIS

mag2lcell [**–m** *mag_dir*] [**–l** *lambda*] [**–T** *tech*] *cellname*

DESCRIPTION

Mag2lcell is a script program, which contains magic's **cif write** and **mag2lif**. *Mag2lcell* reads *cellname*.mag and generates *cellname*.lif.

The following options are recognized:

–m *mag_dir*	Indicates that *cellname*.mag is in directory *mag_dir*. Current directory is default.
–l *lambda*	Specifys the lambda value times 10. (default *lambda* is "10");
–T *tech*	Specifys the technology of MOS. (default *tech* is "scmos");

EXAMPLE

mag2lcell -l 6 *mycell*

SEE ALSO

mag2lif(1),
FUSION: A New MOSIS Service,
Berkeley CAD Tools User's Manual

NAME

mag2lif – generate *.lif* cell for fusion_spc from *.mag* file

SYNOPSIS

mag2lif

DESCRIPTION

Mag2lif is an interactive program, and it reads *cellname*.mag to generate *cellname*.lif.

EXAMPLE

```
$mag2lif
lambda=? 1.0
technology=? cmosn
file_names? addca

<< running magic, message deleted >>

Doing ... addca.mag ...
East/West: Vdd Current (default=0.0 mA) =? 0.06
East/West: GND Current (default=0.0 mA) =? 0.06
North/South: Vdd Current (default=0.0 mA) =?
North/South: GND Current (default=0.0 mA) =?
North only Vdd Current (default=0.0 mA) =?
South only GND Current (default=0.0 mA) =?
East only Vdd Current (default=0.0 mA) =?
East only GND Current (default=0.0 mA) =?
West only Vdd Current (default=0.0 mA) =?
West only GND Current (default=0.0 mA) =?
Done ... addca.lif ...
```

SEE ALSO

FUSION: A New MOSIS Service

AUTHOR

Shih-Lien Lu (USC/ISI)

NAME

mag2rcell – generate macro cell for RNL/IRSIM and VPNR netlist from *.mag* file

SYNOPSIS

mag2rcell [**–m** *mag_dir*] [**–l** *lambda*] [**–v** *vpnr_dir*] [**–T** *tech*] [**–f** [*leading_char*]] [**–F**] *cellname*

DESCRIPTION

Mag2rcell is a script program, which contains magic's **extarct, ext2sim,** and **sim2cell.** *Mag2rcell* reads *cellname*.mag, and generates *cellname*.cell and *cellname*.vpnr.

The following options are recognized:

–m *mag_dir*	Indicates that *cellname*.mag is in directory *mag_dir*. Current directory is default.
–l *lambda*	Specifys the lambda value times 10. (default *lambda* is "10");
–v *vpnr_dir*	Puts VPNR macro cell generated from *mag2rcell* in *vpnr_dir*. Current directory is default.
–T *tech*	Specifys the technology of MOS. (default *tech* is "scmos");
–f *leading_char*	Indicate the leading character of the labels of feedthru lines. If *leading_char* is not indicated, 'F' will be the default value.
–F	List the labels of feedthru lines in the netlist, otherwise, feedthru labels are not listed.

EXAMPLE

mag2rcell -l 6 *mycell*
mag2rcell -m ../mag -v ../vpnr AND

SEE ALSO

sim2cell(1),
Berkeley CAD Tools User's Manual

NAME

prefus – generate fusion_spc and fusion library from NETLIST+

SYNOPSIS

prefus [**–l***lambda*] [**–t***tech*] [**–m***bool*] [**–pl***pad_lib*] [**–pf***pad_frame*] [**–i***val*] *name*

DESCRIPTION

Prefus reads *name*.fnet and generates FUSION specification *name*.spc and *name*.lib.

The following options are recognized:

–l*lambda*	Specifys the lambda value. (default *lambda* is "10");
–t*tech*	Specifys the technology of MOS. (default *tech* is "SCMOS");
–m*bool*	Specifys the way of routing. If *bool* is "true", it means manhattan routing only. If *bool* is "false", then 45 degree routing is allowed;
–pl*pad_lib*	Specifys the pad_library used in the design of chip. (default *pad_lib* is ???.) The pad_libraries available at FUSION are shown in the appendix of NETLIST+;
–pf*pad_frame*	Specifys the pad_frame used in the design of chip. (default *pad_frame* is ???.) The pad_frame available for each pad_library at FUSION are also shown in the appendix of NETLIST+;
–i*val*	Specifys the number of inputs from the each input pad. (default *val* is "2"). If *val* is **1** There is only one node supplied by each input pad. **2** There are two nodes, which are *in* and *in_bar*, supplied by each input pad. **3** There are three nodes, which are *in* and *in_bar* and *in_un*, supplied by each input pad. The number of nodes for each pad_library are shown in the appendix of NETLIST+.

EXAMPLE

prefus -plSLLU2_PADS -pf40PC22X22 -i2 *fuse*

SEE ALSO

FUSION: A New MOSIS Service
Pad library at FUSION in the appendix of NETLIST+

AUTHOR

Tzyh-Yung Wuu (USC/ISI)

NAME

prenet – generate RNL/IRSIM and VPNR netlist from NETLIST+ from *.mag* file

SYNOPSIS

prenet [**–v**] *name*

DESCRIPTION

Prenet reads *cellname*.fnet, and generates RNL netlist.

The following options are recognized:

–v	Generate netlist v_*name*.net for VPNR.

EXAMPLE

prenet -v *fuse*

SEE ALSO

VLSI Design Tools Reference Manual, Release 3.2,
VPNR: Vanilla Place aNd Route System

AUTHOR

Tzyh-Yung Wuu (USC/ISI)

NAME

sim2cell – generate RNL/IRSIM and VPNR macro *.net* from *.sim* file

SYNOPSIS

sim2cell *infile outfile* [–v [*vpnr_dir*]] [–f [*leading_char*]] [–F]

DESCRIPTION

Sim2cell reads .sim file, *infile*, and generates macro cell, *outfile*.

The following options are recognized:

–v *vpnr_dir*	Generate VPNR macro cell in current directory if there is no *vpnr_dir* indicated. Otherwise the VPNR macro cell generated will be put in the directory *vpnr_dir*.
–f *leading_char*	Indicate the leading character of the labels of feedthru lines. If *leading_char* is not indicated, 'F' will be the default value.
–F	List the labels of feedthru lines in the netlist, otherwise, feedthru labels are not listed.

EXAMPLE

sim2cell *AND.sim AND.cell* -v *../vpnr*

SEE ALSO

Berkeley CAD Tools User's Manual
VLSI Design Tools Reference Manual, Release 3.2,

AUTHOR

Tzyh-Yung Wuu (USC/ISI)